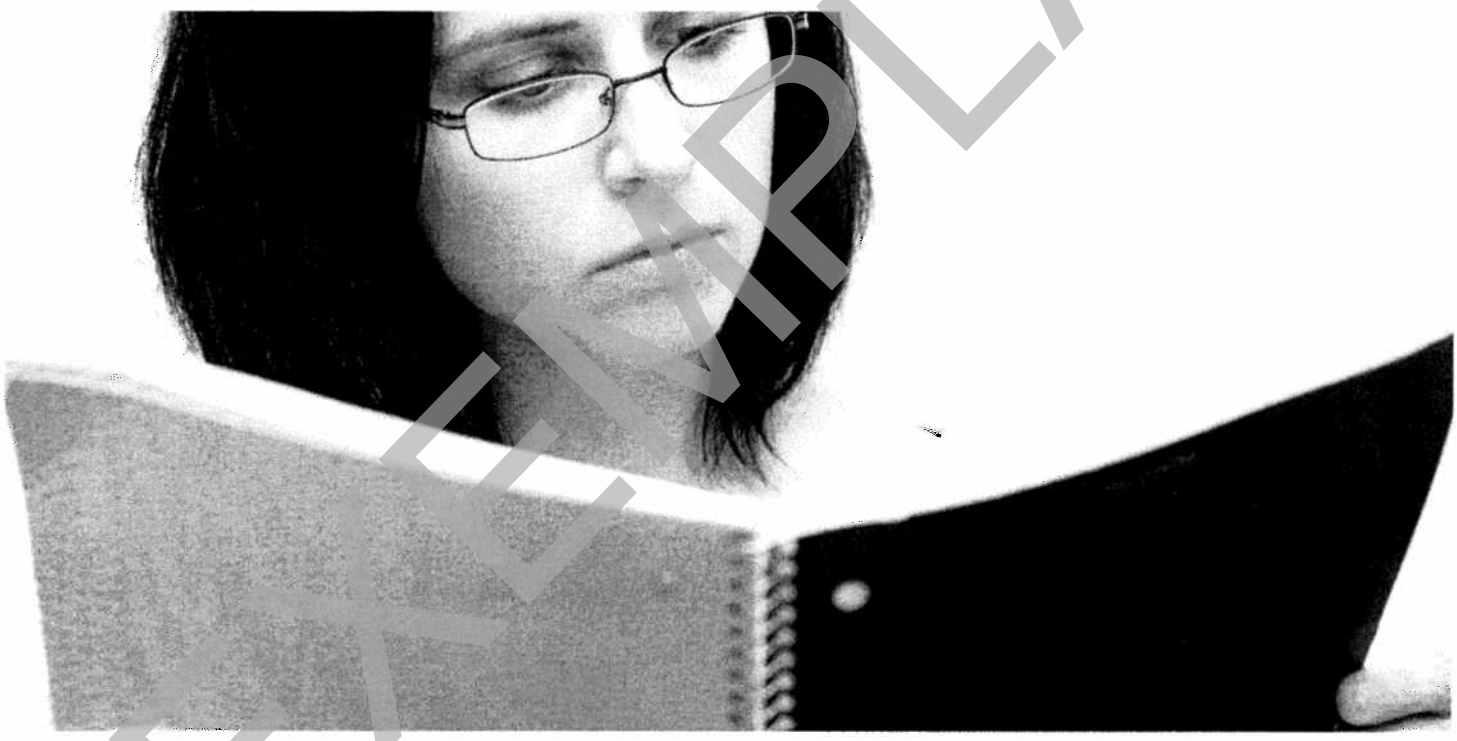
Exemplar work

GCE Computing
COMP4



Candidate 4

Please note: every effort has been made to respect the right of anonymity of the source of this project and of those who have contributed to it, without compromising the work or invalidating it for the purpose to which it is to be put. Some personal details of contributors may therefore remain. We ask the reader to respect their right to privacy.

COMPUTER REPAIR MANAGEMENT SYSTEM

Note: The Teacher's comments are shown in yellow boxes throughout.

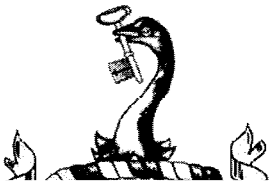
A2 Computing Coursework
APRIL 2010

Contents

Pages appearing in the Appendix are preceded by the letter "A".

Analysis

BACKGROUND



The E Schools' employ around six technicians to deploy and maintain various computing systems around both the Boys' and Girls' Schools. Their remit is wide-ranging, covering access to the internet from workstations within the Schools to repairs to these workstations. Currently, there is no well-structured, organised and informative system for keeping track of issues that arise with the systems installed within the Schools, and the Schools' administrative departments.

Some background details.

IDENTIFICATION OF PROBLEM

There is currently no well-structured, organised and informative system for keeping track of issues that arise with the systems installed within the Schools, and the Schools' administrative departments. Technicians currently use a paper notepad and pen to keep track of an issue, and enter various details about the reported issue into the notepad in a tabular form. An example of the form used is included in appendix page A1. Once an issue has been resolved successfully, it's entry in the notepad is crossed through. Each issue is also given a priority from 1 through to 5 to show how important it is that each issue is resolved: the higher the number, the more important the issue is. This priority is defined by the technician looking at the request, and so some discrepancies may occur when one technician views a job to be more important than another.

Several problems arise from this current system. To begin with, the system for recording jobs is open to misuse: the same job could be entered into the notepad multiple times, for example, and some issues may also be forgotten about if the notepad is not checked over with a relatively high degree of caution. Some issues are also not recorded properly – or at all – within the notepad: this can lead to unnecessary delays for issues to be resolved. In addition, if a sheet of paper is lost, for example, there are no backup copies from which to start working from, leading to unnecessary delays.

Adequate description of the genuine (!) current system. See also appendix A1, interview page 2.

PROSPECTIVE USERS

First and foremost, the system needs to be accessible by all technicians who deal with repairs and issues with current systems. Technicians will have a reasonably advanced knowledge of the systems that they have installed themselves, or have had installed, within the Schools. As such, the system that will be built to keep track of the repairs should not be beyond the technical experience that the technicians have gained.

Generally, the system will not need to be accessed by teaching and administrative staff. There will be no way to log an issue remotely from within the network, as the system will be built in Pascal. If, however, the system was to be built with a web-based language, such as ASP .NET or PHP, then assigning access rights based upon user would be an issue which would need to be looked into.

Satisfactory identification.

USER NEEDS AND ACCEPTABLE LIMITATIONS

- The system needs to be able to record all the details that are currently stored about an issue, as without this, the current system of notepad and pen would have obvious significant advantages over the proposed new system. The details recorded will include:
    - Forename and surname of the person reporting the issue
    - Department which the reporting party belongs to
    - The date that the issue was reported (generally today's date)
    - The name and location of the system at fault
    - A rough description of the issue at fault, and a basic two to three word description of the issue within a specific Category field
- The system needs to be able to record specific details that are currently stored about an asset. In terms of the system, an asset can be defined as any peripheral or system which the school manages. Examples include a workstation, server, projector, interactive whiteboard, and printer. The details recorded will include:
    - Manufacturer
    - Model
    - Location
    - Purchase Date
    - Purchase Retailer
    - Date that warranty expires on
- It should be possible to view the details of one asset or request quickly and easily. This information should be presented in a clear, legible way which is easy to understand quickly.
- The system needs to have a way to track the status of a request as it is completed. The tracking system needs to have an easy-to-understand layout which provides information in a quick and easy-to-read format.
- The system needs to have a way to view statistics about the current number of jobs and assets kept on file. This information should be stored securely, so that only certain people have access to it.
- The files produced by the system must be able to be backed up by the system, and the user should be able to customise where exactly the files are backed up to. In addition, the system should be able to recover data from a file that is imported back into it's directory when data loss has occurred.
- There is no real need for the system to be able to be accessed over a network at this stage: the system can be kept running in the background on one machine, which could be kept in a prominent position so that users can access it quickly.
- The system should be simple-to-use, and – where possible – should run on a wide variety of common operating systems. This will ensure that the system can fit into currently deployed systems, rather than having to deploy systems to fit in with the system itself.

Clear description of user needs.

INTERVIEW

What kind of hardware and software is managed by the IT technicians? We manage all computers, projectors, interactive whiteboards, printers and servers in the School. In terms of software, we manage all aspects of the School's Windows Server, and Citrix thin client servers, as well as various educational packages used by various departments around the School. In addition, we take responsibility for the upkeep of SIMS – the School's pupil record-keeping system.

Interview authenticated.

How do you decide which hardware and software to buy? We have no preferred manufacturer for hardware and software, and so base buying decisions around three main ideas: the specification of the product in question, how it will integrate with the School's existing systems, and its value for money. When we need to buy lots of a particular product, we are often able to qualify for some form of education or bulk purchase discount, as was the case with the thin-client computers spread around the School. Although we have no preferred manufacturers, we do keep a list of preferred resellers and retailers, who we buy our systems through.

Okay, as far as servicing is concerned, do the technicians service everything within School, or are some things serviced through outside contracts? For the most part, as far as possible, all desktop and notebook computers are serviced "in-house". Where a warranty exists on a particular machine, we take full advantage of the warranty and send the machine to the repair centre to be repaired under warranty. The servers are managed and repaired by an external contractor, with whom the School has a maintenance contract.

How is information on job requests currently kept? Currently, information on job requests is kept in a notepad, and certain details about each request are recorded by technicians once the fault is reported by a member of teaching staff. You can have a blank copy of the repair requests sheet, if it's helpful to you?

What information do you currently keep? Currently, the following information is recorded by the technicians:

- The date the request was reported by a staff member
- The location of the machine with the fault, along with the system's name
- Information about the fault
- A priority level (low, medium, high)
- Any other comments

Could you please walk through what exactly happens when a fault is reported? The technician notes the details given above, and assures the person reporting the job that the job will be looked into shortly. At the end of each day, a technician looks through the list and prioritises each request on a scale of Low, Medium and High. The priorities are allocated once other jobs have been taken in to account. For example, we can't have a job about not being able to log into the network being below not being able to use a certain program.

If a job is completed, and new parts need to be sourced from outside, how are these parts obtained? There is one general budget for IT repairs within the school, and any parts and maintenance contracts needing to be sourced from outside. As far as the sourcing of products is concerned, we find the particular product we need – where possible, using parts specifically manufactured by the same manufacturer as the hardware itself – at the cheapest price we can. Where we can't find parts manufactured by the same manufacturer, we purchase parts recommended by the manufacturer for the hardware we have, and failing that we purchase a compatible part, whether this part be manufactured by the original manufacturer or a third-party.

How much time is spent by the technicians managing repair requests? For the most part, the technicians spend most of their time within the School repairing and fixing systems. This

A very good interview.

doesn't, however, reflect badly upon the hardware which the school provides. Due to the fact that we manage so much hardware across all five sections of the Schools – the kindergarten, the two junior schools, and the two senior schools – then repair requests are extremely common. On average, I'd say we get around fifteen to twenty repair requests per week from all five sections of the schools. This number doesn't include repair requests for the servers, as these are not noted down in the repair notepad, since the Windows server is very good at noticing problems, and the technicians can often correct these before users realise a problem has occurred.

DATA SOURCES AND DESTINATIONS

Data	*Source*	*Destination*
Request details	The system will ask the user for the details of each request once they are entered into the system	The request details are stored in an open requests file, so that other sections of the program can use the data it contains.
Employee details	User	The employee details are stored in an employees file, so that other sections of the program can use the data it contains.
Asset details	User	The asset details are stored in an asset file, so that other sections of the program can use the data it contains.
Statistics	Open requests file, assets file and various global variables	User
Agenda	Open requests file	User

Reasonable, but for new system, not existing one.

DATA VOLUMES

From documents that I have seen, it seems that there are between thirty and forty requests submitted during a typical week, There are between 180 and 200 assets – servers, workstations, projectors, digital whiteboards and other peripherals – in the school. Additionally, there are approximately eighty members of staff working in the school.

Brief but covers key points.

DATA FLOW DIAGRAMS FOR EXISTING SYSTEM AND PROPOSED SYSTEM

See appendix page A2

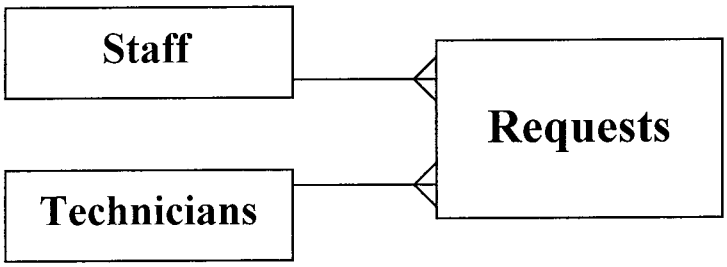
DATA DICTIONARY

Field	*Purpose*
Date	Date the request was reported to the technicians
Location	Location of the faulty system
Fault	Description of the faulty system
Priority	How important the fault request is: low, medium or high
Comments	Any other comments regarding this request

Currently, these are the only details which are recorded for each request. The current data stored about each request is not sufficient, and therefore my system will make allowances for this.

Covers current system recording of faults but not assets.

E-R MODEL



Each staff member can report many requests, yet each request can only belong to one staff member. Similarly, each technician can be assigned to many requests, yet each request can only be assigned to one technician.

Satisfactory. Does not include assets.

OBJECTIVES

I will be attempting to create a system that helps the Schools' IT technicians to organise incoming repair requests in a more efficient way. The system will be able to tell technicians which jobs need completing, and will put them in an order of some sort.

1. The system must allow certain data about each job request to be stored. At the very least, the system needs to be able to store the same data that is currently collected under the existing system, unless there is a good reason to not store the data. This data includes:
    - Full name of the person reporting the issue
    - Department which the reporting party belongs to
    - The date that the issue was reported – generally today's date
    - The name and location of the system at fault
    - A rough description of the issue at fault.
2. The system needs to be able to show the current status of a job upon request.
    a. This should be shown within the Requests menu.
    b. In addition, this should be shown within the Tracking system.
3. The system needs to be easy-to-use and logical for a user to follow.
    a. A basic colour scheme should be followed throughout the system.
    b. Where possible, the system should make use of common keys. For example, D should not be assigned to delete an asset in one area, and confirm an option in another area without good reason.
4. The system needs to work on as many platforms as possible.
    a. There should be no extensive requirements to heavily adapt any existing systems to run this system as intended.
    b. To achieve objective 4A, it should also be noted that the system should be built to run on as many common operating systems as possible.
5. The system needs to be able to produce statistics about any job requests.
    a. The system should be able to show the total number of jobs and assets.
    b. The system should be able to show the total number of jobs assigned to each technician.

Clear - could be extended. Contains adequate scope.

6. Security is important:

a. All files generated by the system should be able to be backed up with minimal intervention from the user.
b. Certain areas of the system may contain sensitive information. As such, it is important that a password can be set to protect this information from prying eyes.

APPRAISAL OF POTENTIAL SOLUTIONS

There are various commercial software solutions available to manage repair jobs, and associated tasks. A quick Google search brings up around twenty nine and a half million results for "computer repair management software". Of course, not all of these are legitimate systems, and there are far more legitimate systems than I have time to compare to my proposed system. For this reason, I will only look at two systems.

BlancheSoft TSMan 2009 is an application that runs within Windows, and does not require any specific software to allow it to run. TSMan is obviously built for the commercial market, whereby repair companies could deploy the solution at a customer-level, and perform various point-of-sale administrative tasks, such as taking payment, stock-taking, refunds and discounts. My proposed system would only be used in-house, and these point-of-sale features would not be required.

SolarVista is another Windows application which, akin to TSMan, requires no additional software for it to run properly. Again, SolarVista is obviously another product intended for commercial markets and SMEs (Small to Medium Enterprises). An accompanying product for mobile terminals allows employees to quickly access information about a particular job whilst they are "in the field".

Brief comparison with two commercial systems.

JUSTIFICATION

Dependent upon the customer's specific needs, specific solutions may be more appropriate. For large-scale customers, and those that need to be able to rapidly share data with field-based employees, SolarVista would be a good solution. Medium-scale customers would find TSMan a perfect solution as it facilitates point-of-sale tasks, allowing for a store to be completely run from its features. My solution will cater for smaller customers, and customers where a point-of-sale system is not a requirement. Additionally, the system will be built in the Pascal programming language, and therefore will be capable of running on most hardware, regardless of hardware specifications. Subject to being able to find someone who knows some Pascal commands, the system can also be adjusted to suit a certain user's requirements. With SolarVista and TSMan, a requirement of a programming language such as Visual Basic would be required, making the customisation possibilities more expensive and more difficult to run.

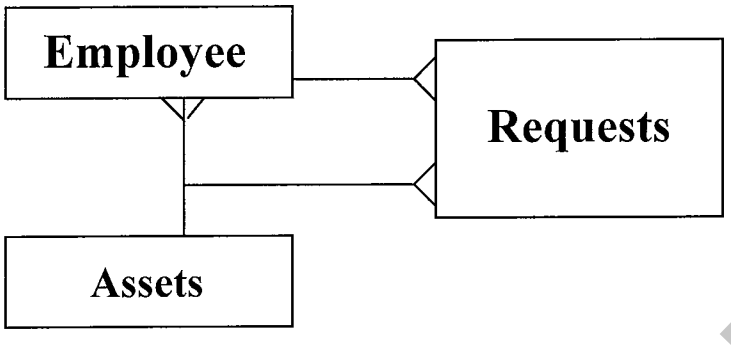
Reasonable justification.

Design

OVERALL SYSTEM DESIGN

See appendix, pages A2 to A8. DFDs.

DATABASE ENTITY RELATIONSHIP MODEL



Each employee can have access to many assets, and can place many requests. An individual asset can belong to many different requests.

Oddly drawn.

DEFINITION OF MODULAR STRUCTURE OF SYSTEM

See appendix, page A7.

DEFINITION OF DATA REQUIREMENTS

Item	***Reason Needed***
Request ID	Uniquely identifies the request. This is required if two requests are entered about the same fault.
Request Date, Year	The date that the request was entered. This is split into three as Pascal's get date command gathers the data in three sections.
Request Date, Month	
Request Date, Day	
User's Forename	The reporting user's forename.
User's Surname	The reporting user's surname.
User's Department	The department that the reporting user belongs to.
System Location	The location of the system that the user is reporting a fault about.
System Name	The name of the system that the user is reporting a fault about.
Error Category	The category of error that the fault relates to
Error Description	A full description of the error being reported
Current Status	The current status of the report
Notes	Any notes that have been added to further explain the problem
Assigned Technician	The technician assigned to the job.

Most of this information is typed in when a new request is added to the system. The Request ID is automatically assigned by the system, and the three parts of the request date are calculated automatically by the system. The date is gathered through a DOS command. The Current Status, Notes and Assigned Technician fields are completed through the Update Request function.

Item	***Reason Needed***
Asset ID	A unique number assigned to each asset. This is required if two assets of the same manufacturer, model etc. exist and there is no way to differentiate between them.
Manufacturer	Records the manufacturer of the asset in question.
Model	Records the model of the asset in question.
Location	Records the location of the asset in question.
Purchase Date	Records the purchase date of the asset, if this can be recorded. Examples of times when this field may appear blank include when the purchase date is indeterminable due to a lost receipt, for example.
Purchase Retailer	Records where the asset was purchased from. This could be useful if any qualms with the asset arise.
Warranty Expiration Date	Records when the warranty for this asset expires.

With the exception of the Asset ID, all information is typed in manually when an asset is added to the system. The Asset ID is generated automatically and given to the user in the summary screen.

Detailed data dictionary.

USER INTERFACE

The user interface will be designed around a simple menu. All of the features of the system are quickly accessible through the menu, by choosing a numerical option assigned to each feature. It is important that the user interface is of consistent design:

- Each screen has a black background, and features dark grey text. This makes the text legible quickly, and no strain is placed upon the eyes.
- Headings are at the top left of each page. This allows the user to quickly realise which page they are currently looking at.
- Menu options are always numerical. This allows a certain degree of consistency across the entire program.
- Errors and serious warnings are displayed in red upper-case text. Standard-level warnings are displayed in yellow text. This draws attention the warnings and errors, as the text is a different colour to that of which the text usually appears in.

Good rationale. Several screens planned with just satisfactory level of explanation.

This is the main menu screen, and appears when the program is first launched. It allows easy access to the program's main features, and is set out in a clear and concise way. For example, all tasks related to the management of requests are listed in the Requests option. Popular options are listed nearer the top of the menu list, making them easily visible.

```
CURRENT STATISTICS

Currently, there are 6 requests in the system, of which:
      - 2   jobs are deemed open
      - 2   jobs are under investigation
      - 0   jobs are awaiting parts
      - 0   jobs are being resolved
      - 0   jobs require attention
      - 0   jobs are ready to be collected
      - 2   jobs are deemed as Other

AB currently has 4 jobs assigned.
CD currently has 1 jobs assigned.

The last request was submitted on: 03/03/2010.
```

The statistics screen displays statistics about the currently open jobs, and allows quick and instant access to important and useful pieces of information about requests. The data provided will span the entire system, not just one particular job. This information will be useful for managers to be able to see where exactly a request is currently at. The statistics screen will list the number of requests currently categorised under each category, and will also list the number of requests assigned to each technician. Finally, the screen lists the date of which the last request was submitted on.

```
TRACKING

This service allows you to check the status of any open or archived requests.
You can view:
      - Request details
      - Status updates
      - Technician's notes

A tracking reference is required to use this service.
Tracking references are available once you have submitted a request.
It is NOT possible to relocate a tracking reference if it is misplaced.

Tracking Reference:

To quit, press 0 on the numpad, then hit [ENTER].
```

The tracking screen allows access to the tracking system. It explains what the service can provide, and what is required to use the service. The explanatory notes are given before the user enters a tracking reference, so that the user is more likely to understand what exactly the service can provide.

```
AGENDA

#  DATE      S  NAME          DEPT    LOCATION     CATEGORY     DESCRIPTION
2  10/2/10   1  D Ferguson    Comp.   LK (LK-05)   Email        No email
3  10/2/10   2  S Harvey      HM      HM (HM-01)   Printer      Cannot print
4  10/2/10   4  P Meakin      Comp.   LL (LL-01)   Monitor      "No input"
5  10/2/10   3  A Christian   Hist.   LJ (LJ-01)   Software     Install Word
6  10/2/10   5  M Hone        Hist.   LH (LH-03)   Email        Cannot send email

Hit [ENTER] to return to the main menu.

Status Glossary
 1 : Open
 2 : Under investigation
 3 : Awaiting parts
 4 : Being resolved
 5 : Requires attention
 6 : Other
 7 : Closed
```


The agenda screen allows technicians to instantly see what jobs are open, and which need fixing as a priority. The screen shows information about each job in a tabular form, and provides a Status Glossary, as the request statuses are listed in a numerical order, rather than as the full text. A quick description of the error assigned for each request is given. Each technician looks up his or her own jobs, and so only those jobs assigned to a particular technician are shown.

The Add New Request screen allows users to enter data about faults directly in to the system. It only asks for necessary information, as this will allow for more productive use of a technician's time. The date is generated automatically by the program by consulting the computer's clock.

SAMPLE OF PLANNED DATA CAPTURE AND ENTRY

See appendix, page A8.

IDENTIFICATION OF VALIDATION REQUIRED

Procedure	Data Item	Check Type	Check Details
Check Password	Entered Password	Value Check	The entered password must be equal to the currently assigned password.
Add Request	Forename	Length Check	The forename needs to be less than twenty characters in length
Add Request	Surname	Length Check	The forename needs to be less than twenty characters in length
Add Request	Date	Format Check	Value is in DD/MM/YY format.
Find Asset by Type	Type Number	Value Check	Value is within acceptable range – ie 1 to 6
Add New Asset	Location	Length Check	Length is higher than two characters long
Set Password	Existing Password	Value Check	Value actually exists! ie – existing password isn't blank, using a length check: length of old password is more than 1.
Set Password	New Password	Value Check	New password is the same value as new password verify
Tracking	Tracking Reference	Value Check	Check tracking reference exists
Agenda	Technician's initials	Length Check	Check initials are three characters long
Various menus throughout system	Exit number	Presence check	Check that the quit number hasn't been called – eg 0 from the main menu quits the program, so check 0 hasn't been entered by the user.

SAMPLE OF PLANNED DATA VALIDATION

Check	Check Data	Expected Result
Check newPassword is the same as newPasswordVerify	newPassword = 12345 newPasswordVerify = 12345	newPassword and newPasswordVerify match in terms of value. New password is accepted.
Check asset location is two characters long.	184959	System should prompt the user to re-enter the location. Will keep repeating until location is two characters long.

good suggestions for validation, a range of checks.

Good structures for 4 files.

FILE ORGANISATION, RECORD STRUCTURE AND PROCESSING

Field Name	Type	Size	Notes
reqID	Integer	2	Unique ID which identifies each request. This is the primary key.
reqDateYear	Integer	2	These fields are used when the program goes to the operating system to get the current date.
reqDateMonth	Integer	2	
reqDateDay	Integer	2	
reqUserForename	String	10	The forename for the user reporting this job.
reqUserSurname	String	10	The surname for the user reporting this job.
reqUserDept	String	15	The department which the user reporting this job belongs to.
reqSystemLoc	String	5	The location of the system which the user is reporting a fault about.
reqSystemName	String	5	The name of the system which the user is reporting a fault about.
reqErrorCat	String	100	A broad categorical description of the fault which the user is reporting.
reqErrorDesc	String	255	A further, in-depth description of the fault which the user is reporting.
reqCurrentStatus	Integer	1	The current status of a reported job, in numerical form.
reqNotes	String	255	Notes associated with this reported job.
reqAssignedTech	String	3	The initials of the technician assigned to this request.

The above details are recorded in a file named *OpenReq.dat*. This file is stored in the same directory as the system itself is. The records are stored in order of reqID, which is the record's primary key. As the records are all of a fixed length, this file can be accessed using direct access techniques. When a user wants to find a request based upon it's ID, the record can be moved to directly as reqID is stored in numerical order. However, when a user wants to find a request based upon surname, for example, the system must search through each record sequentially until it finds the record matching the surname entered by the user. When a user adds a new request to the system, the details are appended to the bottom of the file. This data is then stored in order of reqID, so that the data can be looked through quickly. When a request is deleted, the reqID is set to 0 and the system overlooks that request.

Good comments on file processing.

Field Name	Type	Size	Notes
astID	Integer	2	Unique ID associated with this asset.
astType	String	50	Type of asset – e.g. computer, server, monitor, peripheral etc.
astMfgr	String	50	Asset manufacturer
astModel	String	50	Asset model
astLoc	String	50	Asset location
astPurchDate	String	10	Asset's purchase date in dd/mm/yyyy format
astPurchRetl	String	30	Asset's purchase retailer
astWarrExpir	String	10	Asset's warranty expiration date in dd/mm/yyyy format

The above details are recorded in a file named *assets.dat*. This file is stored in the same directory as the system itself is. The records are stored in order of astID, which is the record's primary key. As the records are all of a fixed length, this file can be accessed using direct access techniques. When a user wants to find an asset based upon it's ID, the record can be moved to directly as astID is stored in numerical order. However, when a user wants to find an asset based upon type, for example, the system must search through each record sequentially until it finds the record matching the type entered by the user. When a user adds a new asset to the system, the details are appended to the bottom of the file. This data is then stored in order of astID, so that the data can be looked through quickly. When an asset is deleted, the astID is set to 0 and the system overlooks that asset. If the asset is then brought back in future, the user must add the asset as if it were brand new using the Add Asset procedure.

Field Name	Type	Size	Notes
allRequests	Integer	5	Notes the total number of requests in the system at any one time.
invesRequests	Integer	5	Notes the total number of requests that are deemed under investigation in the system at any one time.
awaitRequests	Integer	5	Notes the total number of requests that are deemed awaiting collection in the system at any one time.
attenRequests	Integer	5	Notes the total number of requests that are deemed to be requiring attention in the system at any one time.
otherRequests	Integer	5	Notes the total number of requests that are deemed under the "other" categorisation in the system at any one time.

The above details are recorded in a file named *StatsType.dat.* This file is stored in the same directory as the system itself is. There is no primary key in this file due to the fact that it is only written to by one function – the Add Request function - and read from by another – the Statistics function. The data is stored in the order it appears in the table above. When a new

request is added to the system, the value of allRequests is increased by one. This ensures that the allRequests value is always correct. Similarly, when a request is updated, the values of invesRequests, awaitRequests, attenRequests and otherRequests are increased accordingly. When a request is deleted, the current status of the request is decreased by the value of 1 so that statistics produced are as accurate as possible.

Field Name	Type	Size	Notes
LastReqID	Integer	3	Stores the last request ID.
LastAstID	Integer	3	Stores the last asset ID.

The above details are recorded in a file named *IDData.dat.* This file is stored in the same directory as the system itself is. When a new request or asset is appended to the system, the LastReqID or LastAstID fields are updated with the new last request or asset ID. This is used by the system to make sure that no two assets or requests have the same ID, even if the program is closed and re-opened.

IDENTIFICATION OF STORAGE MATERIAL AND FORMAT

As a package, my system will rely on five data files and the program itself:

- OpenReq.dat
- Assets.dat
- Employees.dat
- StatsType.dat
- IDData.dat
- Project.pas

Obviously, the files suffixed in the .dat extension increase in size as more data is added to them through the Pascal program. Saying this, however, with a modest number of records in each file, I don't expect the size to grow above 1mb. With ten records in each file, the package is a total of 45kb, including the Pascal file which will be about 35kb. As such, it would be feasible to use a storage medium as small as a floppy disc, or as large as a hard disc drive. Of course, there are certain disadvantages to using certain storage mediums. Floppy discs, and floppy disc drives, are not as readily available as flash drives and hard disc drives. Floppy discs are now deemed to be a rather old, and fairly useless storage medium, but floppy disc drives do still exist on many older computers. For these reasons, it would be practical to deploy this system on a floppy disc, however it would not be recommended if there are other mediums available. As the system is not deployable over a network, there is no need to deploy the system on to a storage medium connected to a network. With some alterations, however, the system could be made to work over a network.
From an economical position, hard disc drives are relatively cheap to purchase. As a result, it would be recommended that the user would run the system from a hard disc drive. The little space that the entire system is projected to use would mean that the user is able to use their system for other purposes as well, if required.

Clear justification.

Open Requests file

This file will be accessed using direct access as information about the requests submitted will be saved directly to the file at the point of entry. This means that tape is not a suitable storage medium for these files.

Request Record Size:	668 bytes
Estimated Number of Requests:	100 simultaneously co-existing
File Size:	668 x 100 = 65.2kb (at 1024 bytes per kb)

Assets file

This file will be accessed using direct access as information about the requests submitted will be saved directly to the file at the point of entry. This means that tape is not a suitable storage medium for these files.

Request Record Size:	252
Estimated Number of Requests:	200
File Size:	252 x 200 = 49.2kb (at 1024 bytes per kb)

Employees file

This file will be accessed using direct access as information about the requests submitted will be saved directly to the file at the point of entry. This means that tape is not a suitable storage medium for these files.

Request Record Size:	82
Estimated Number of Requests:	100
File Size:	82 x 100 = 8kb (at 1024 bytes per kb)

StatsType file

This file will be accessed using direct access as information about the requests submitted will be saved directly to the file at the point of entry. This means that tape is not a suitable storage medium for these files.

Request Record Size:	10
Estimated Number of Requests:	100
File Size:	10 x 100 = 0.97kb (at 1024 bytes per kb)

IDData file

This file will be accessed using direct access as information about the requests submitted will be saved directly to the file at the point of entry. This means that tape is not a suitable storage medium for these files.

Request Record Size:	4
Estimated Number of Requests:	200
File Size:	4 x 200 = 800 bytes

IDENTIFICATION OF SUITABLE ALGORITHMS FOR DATA TRANSFORMATION

Tracking procedure A linear search.

```
Start procedure "viewTracking"
Define TrackingRef, searchPosition and subMenuOption variables as integers
Define foundRecord variable as Boolean
Clear entire screen
Write blank line
Write "tracking" header
Write blank line
Write "This service allows you to check the status of any open or archived requests"
Write "You can view:"
Write "- Request details"
Write "- Historical status updates"
Write "- Engineer notes"
Write blank line
Change text colour to yellow
Write "A tracking reference is required to use this service."
Change text colour to light grey
Write "Tracking references are available once you have submitted a request"
Write "It is NOT possible to relocate a tracking reference if misplaced."
Write blank line
Write blank line
Write "Tracking reference: "
Write blank line
Write "To quit, press 0 on the numpad, then hit [ENTER]."
Go to position 23, 14 on the display
Set value of searchPosition to 0
Set value of foundRecord to false
Read value entered into TrackingRef
Write blank line
Assign open requests file using directories specified at start of program
Reset open requests file, ready for next use
Whilst not end of open requests file do
```

Use of serial access.

```
    Begin
    Read request details from openreqfile,newreq
    If entered reqId equals value of TrackingRef then
        Begin
        Set foundRecord to true
        Clear screen
        Write blank line
        Write "Tracking options for request ", write tracking reference
        Write blank line
        Write "#       Action"
        Write blank line
        Write "--------------------"
        Write "1       Display notes and status"
        Write "2       Display original request details"
```

```
                Write "--------------------"
                Write "Opt: "
                Read subMenuOption
                If subMenuOption = 2 then call FindRequestById procedure
                Else
                        Clear screen
                        Write blank line
                        Write "TRACKING"
                        Write blank line
                        Write "Tracking Ref: "
                        Write request ID
                        Write blank line
                        Write blank line
                        Write "Current Status: "
                        Put newreq.reqCurrentStatus into CurrentStatus procedure
                        Write CurrentStatus value
                        Write blank line
                        Write "Request Notes: "
                        Write value of newreq.reqValue
                        Write blank line
                        Write blank line
                        Write "Assigned to: "
                        Write value of newreq.reqAssignedTech
                        Write blank line
                        Write "Press [ENTER] to return to main menu"
                        Read line
        End while
If FoundRecord = false, then
        Clear screen
        Write blank line
        Write "TRACKING"
        Write blank line
        Change text colour to red
        Write " ERROR: "
        Change text colour to light grey
        Write "No tracking reference has been found!"
        Write blank line
        Write "Are you sure this tracking reference exists?"
        Write blank line
        Write "Hit [ENTER] to continue"
End If
Close open requests file
Read line
```

Agenda procedure

```
Start procedure "agenda"
Define enteredInitials variable as string with value limit of 5
Define currentJobs as an array of 1 to 20 of reqType
Define a, b, j, jcount, x and y as integers
```

```
Define temp variable of type reqType
Set value of X to 0
Clear screen
Write blank line
Write "AGENDA"
Write blank line
Write "Enter initials: "
Read entered value for previous line into enteredInitials
Set value of jCount to 0
Assign open requests file
Reset open requests file, ready for next use
While not end of open requests file, do
        Open request record
        Write "*", followed by assignedTech, CurrentStatus and reqId
        Read line
        If:     entered initials equal reqAssignedTech
                reqCurrentStatus = 7
                and newreq.reqID = 0

        then
                Set value of jCount to jCount+1
                Set newreq to value of currentjobs after jCount value has been entered
        End If
Close open requests file
Read line

If value of jCount is more than 0, then
        Go to position 1, 2 on screen
        Set value of y to 4
        Go to position 1, y on screen
        Clear screen
        Set value of a to 1
        For a to jcount-1 do
                Set value of b to 1
                For value of b to jcount-1 do
                        Place value of b into CurrentJobs.
                        If value of reqCurrentStatus is more than the value of CurrentJobs + 1
                        then    Set variable temp to value of CurrentJobs
                                Add 1 to the value of CurrentJobs
                                Set value of temp to CurrentJobs+1
End If

Set value of j to 1
For j=1 to value of jcount do
                Go to position 2, 2 on screen
                Write "#"
                Go to position 1, y on screen
                Write value of currentJobs.reqID
                Go to position 6, 2 on screen
                Write "DATE" as header
```

Counting jobs!

```
                Go to position 5, y on screen
                Write value of currentJobs.reqDateDay, reqDateMonth, reqDateYear mod 100
                Go to position 14, 2 on screen
                Write "S" as header
                Go to position 13, y on screen
                Write value of reqCurrentStatus
                Go to position 16, 2 on screen
                Write "NAME" as header
                Go to position 15, y on screen
                Write value of reqUserForename (truncated to 1 character), reqUserSurname
                Go to position 28, 2 on screen
                Write "DEPT" as header
                Go to position 27, y on screen
                Write value of reqUserDept
                Go to position 35, 2 on screen
                Write "LOCATION" as header
                Go to position 34, y on screen
                Write value of reqSystemLoc, reqSystemName
                Go to position 48, 2 on screen
                Write "CATEGORY" as header
                Go to position 47, y on screen
                Write value of reqErrorCat
                Go to position 58, 2 on screen
                Write "DESCRIPTION" as header
                Go to position 57, y on screen
                Write value of reqErrorDesc
        Y:=Y+1
End For

Else
        Write blank line
        Write blank line
        Write "Hit [ENTER] to return to main menu."
        Read line.
```

DESCRIPTION OF MEASURES PLANNED FOR SECURITY AND INTEGRITY OF DATA

Validation

Data will be checked according to various rules implemented within the code, as listed in an earlier section. By allowing validation to happen when the data is input, it can be checked so that the data stored in the files is as integral as possible when it is saved. This reduces the risk of corrupt, or unreadable, data appearing at later stages.

Backups

Backups will not be automatically managed by the system itself. Instead, a robust backup system is strongly recommended. As this system is intended to be installed in major organisations, there is an assumption that a large organisation will have some form of backup strategy in place. The files generated by the systems are not particularly large in terms of file sizes. As a result, backing up the system's files will not be a time-consuming or expensive measure.

File access

Currently, my program is only intended to be used by technicians, and therefore would not be used by staff members themselves. For this reason, I have decided not to implement a username/password system for the entire system. However, there is a password system which is called whenever certain procedures and functions are used. These procedures and functions are generally only those which show sensitive data, such as an agenda. Only one password can be set for the entire system, as its primary purpose is to protect certain data from staff members, not to protect data from other technicians. Technicians, therefore, share the same password, which can be changed at any time from the Settings menu.

Very good.

OVERALL TEST STRATEGY

It is extremely important that the system can perform the tasks expected of it, without any fundamental errors. As such, it is crucial that the system performs well in all kinds of scenarios. I will be testing my system in three types of scenarios – under "typical" conditions, in "extreme" conditions – where the data entered is only just on the verge of being accepted, and in "erroneous" conditions – where any data entered should not be accepted.

I will test all parts of my system, including:

- Menus and navigation
- Requests features
- Agenda features
- Statistics features
- Assets features
- Features related to data handling, and how data is passed between procedures and functions

A limited description.

```
{ Computer Repair Management System }
{ Version 1.0                       }
{ Last update: 27/03/10             }
{ (C) 2010, Andrew Langhorn         }

Program jobs;

Uses    crt,dos;

Const   openreqfilename   = 'c:\openreq.dat'   ; {location of
request data}
        employeefilename  = 'c:\employees.dat' ; {location of
employee data}
        assetsfilename    = 'c:\assets.dat'    ; {location of asset
data}
        idfilename        = 'c:\iddata.dat'    ; {location of
request id data}

Type    reqtype = Record
  reqID             : integer     ; {request ID}
  reqDateYear       : integer     ; {request date year}
  reqDateMonth      : integer     ; {request date month}
  reqDateDay        : integer     ; {request date day}
  reqUserForename   : string[10]  ; {request user forename}
  reqUserSurname    : string[10]  ; {request user surname}
  reqUserDept       : string[15]  ; {request user dept}
  reqSystemLoc      : string[5]   ; {request system location}
  reqSystemName     : string[5]   ; {request system name}
  reqErrorCat       : string[100] ; {request error category}
  reqErrorDesc      : string[255] ; {request error description}
  reqCurrentStatus  : integer     ; {request current status}
  reqNotes          : string[255] ; {request notes}
  reqAssignedTech   : string[3]   ; {technician assigned to request}
End;

Type    emptype = Record
  empId           : integer     ; {employee id}
  empForename     : string[20]  ; {employee forename}
  empSurname      : string[20]  ; {employee surname}
  empDept         : string[20]  ; {employee department}
  empPhone        : string[20]  ; {employee phone number}
End;

Type    asttype = Record
  astId           : integer     ; { ID }
  astType         : string[50]  ; { type }
  astMfgr         : string[50]  ; { manufacturer }
  astModel        : string[50]  ; { model number }
  astLoc          : string[50]  ; { room name }
  astPurchDate    : string[10]  ; { date dd/mm/yyyy}
  astPurchRetl    : string[30]  ; { purchase retailer }
  astWarrExpir    : string[10]  ; { date dd/mm/yyyy }
End;

Type    statstype = Record
```

5 user-defined data types.

```
  AllRequests     : integer          ; { Total number of requests in
system }
  InvesRequests   : integer          ; { Total number of requests
being investigated in system }
  AwaitRequests   : integer          ; { Total number of requests
awaiting attention in system }
  AttenRequests   : integer          ; { Total number of requests
requiring attention in system}
  OtherRequests   : integer          ; { Total number of requests as
Other in system }
End;

Type    assigntype = Record
  assignedtech    : string[3]; { Technician initials }
  jobcount        : integer;   { Number of jobs allocated to tech.
defined above }
End;

{ Various variables appear below to ensure the system }
{ operates smoothly, and to allow data to flow easily }
{ between procedures, functions and files             }

Var     openreqfile          : file Of reqtype          ;
        { defines openreqfile to be defined under the reqtype
structure }
  employeesfile        : file Of emptype          ;
        { defines employeesfile to be defined under emptype
structure }
  assetsfile           : file Of asttype          ;
        { defines assetsfile to be defined under asttype structure }
  idfile               : file Of integer          ;
        { defines idfile to be defined under idtype structure }
  newreq               : reqtype                  ;
        { defines each new request to be defined within reqtype }
  newemp               : emptype                  ;
        { defines each new employee to be defined within emptype }
  newast               : asttype                  ;
        { defines each new asset to be defined within asttype}
  stats                : statstype                ;
        { defines statistics within statstype }
  latestreqID          : integer                  ;
        { stores last-used request ID }
  latestastID          : integer                  ;
        { stores last-used asset ID }
  GlobalTotalJobsCount : integer                  ;
        { counts total number of current jobs }
  jobcountbystatus     : array[1..7] Of integer   ;
        { separates GlobalTotalJobsCount by status }
  jobcountbytech       : array[1..10] Of assigntype ;
        { separates GlobalTotalJobsCount by technician }
  y,m,d,dow            : word                     ;
        { stores current date }
  techcount            : integer                  ;
        { total number of technicians in system }
  globalpassword       : string                   ;
        { stores password for use in certain scenarios }
```

Quite a lot of global variables.

```
{ The following procedure writes the last created     }
{ Request ID (latestreqID) and Asset ID (latestastID }
{ and saves them to a text file. This text file is    }
{ then referenced when a new request is made, to      }
{ make sure that no two requests or assets are given }
{ the same ID.                                        }

Procedure saveids;
{ var globally declared }
Begin
  assign(idfile,idfilename);
  rewrite(idfile);
  write(idfile,latestreqID);
  write(idfile,latestastID);
  close(idfile);
End;

{ The following procedure is used to enforce password regulations }
{ in certain procedures and functions. It asks for the currently  }
{ set password, and will not allow access to the relevant         }
{ procedure or function until it is set.                          }

Procedure checkpassword;

Var enteredpassword : string;   { stores password entered by user }
Begin
  clrscr;
  writeln(' ');
  textcolor(red);
  writeln(' SYSTEM SECURITY');
  textcolor(lightgray);
  writeln(' ');
  writeln(' This section is password protected.');
  writeln(' To access it, please enter the password you have created
in Settings.');
  writeln(' ');
  write(' Password: ');
  read(enteredpassword);
  If enteredpassword<>globalpassword Then
    Begin
  textcolor(red);
      writeln(' ERROR');
      writeln(' ');
      textcolor(lightgray);
      writeln(' The password you have entered is incorrect.');
      writeln(' Please try again.');
      clrscr;
      checkpassword;
    End
  Else
End;

{ The following procedure uses a case statement }
{ to write up the current status when the       }
{ system is passed a raw integer. This is used  }
```

```
{ by the tracking system later in the program.  }

Procedure showstatus(status:integer);

Begin
  Case status Of
    1: writeln(' Open');
    2: writeln(' Under investigation');
    3: writeln(' Awaiting parts delivery');
    4: writeln(' Being resolved');
    5: writeln(' Requires attention');
    6: writeln(' Ready for collection');
    7: writeln(' Other');
    8: writeln(' Closed');
  End;
End;

{ The following procedure uses a case statement }
{ to write up the correct asset type when the   }
{ system is passed a raw integer. This is used  }
{ later in the program.                         }

Procedure assettype(typenum:integer);
Begin
  Case typenum Of
    1: writeln(' Workstation');
    2: writeln(' Server');
    3: writeln(' Whiteboard');
    4: writeln(' Projector');
    5: writeln(' Peripheral');
    6: writeln(' Other');
  End;
End;

{ The following procedure checks the text file }
{ created by the saveids procedure, and starts }
{ a request with a unique ID. The unique ID    }
{ always follows the last ID recorded in the   }
{ text file produced by saveids.               }

Procedure loadids;
{ var globally declared }
Begin
  assign(idfile,idfilename);
  reset(idfile);
  read(idfile,latestreqID);
  read(idfile,latestastID);
  close(idfile);
End;

{ The following procedure accepts entries from }
{ users regarding new requests. This data is   }
{ then saved in openreq.dat, and is shared     }
{ between procedures which use this data in    }
{ different ways.                              }
```

Parameter passing.

```
Procedure AddNewRequest;
{ var declared globally }
Begin
  clrscr;
  writeln(' ');
  writeln('          ADD REQUEST ');
  writeln(' ');
  write(' Forename:          ');
  readln(newreq.reqUserForename);
  write(' Surname:           ');
  readln(newreq.reqUserSurname);
  write(' Department:        ');
  readln(newreq.reqUserDept);
  GetDate(y,m,d,dow);
  writeln(' Today''s date:      ',d:1,'/',m:1,'/',y:1);
  newreq.reqDateYear := y;
  newreq.reqDateMonth := m;
  newreq.reqDateDay := d;
  write(' System Location:   ');
  readln(newreq.reqSystemLoc);
  write(' System Name:       ');
  readln(newreq.reqSystemName);
  write(' Fault Category:    ');
  readln(newreq.reqErrorCat);
  write(' Fault Description: ');
  readln(newreq.reqErrorDesc);
  newreq.reqAssignedTech := ' ';
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  seek(openreqfile,filesize(openreqfile));
  latestreqID := latestreqID+1;
  newreq.reqID := latestreqID;
  GlobalTotalJobsCount := GlobalTotalJobsCount+1;
  clrscr;
  gotoxy(2,2);
  writeln('THANK YOU ');
  writeln(' ');
  writeln(' Your request has been submitted and saved
successfully!');
  writeln(' You have submitted the following details: ');
  writeln(' ');
  writeln(' Name:              ',newreq.reqUserForename,'
',newreq.reqUserSurname);
  writeln(' Department:        ',newreq.reqUserDept);
  writeln(' Submission Date:
',newreq.reqDateDay,'/',newreq.reqDateMonth,'/',newreq.reqDateYear);
  writeln(' System Name:       ',newreq.reqSystemName);
  writeln(' System Location:   ',newreq.reqSystemLoc);
  writeln(' Fault Category:    ',newreq.reqErrorCat);
  writeln(' Fault Description: ',newreq.reqErrorDesc);
  writeln(' ');
  writeln(' You will need a Request ID to track the status of this
request.');
  writeln(' Your assigned Request ID is: ',newreq.reqId);
  write(openreqfile,newreq);
  close(openreqfile);
```

```
  saveids;
  readln;
End;

{ The following procedure updates the status of an already }
{ existing request. Users provide the procedure with the   }
{ request ID and any other information the procedure       }
{ requires.                                                }

Procedure UpdateRequestStatus;

Var
  EnteredRequestId : integer;              { stores request ID entered
by user }
  RequestStatusUpdateOption : integer;     { stores request status
update }
  AddNotesOption : char;                   { stores answer regarding
notes }
Begin
  clrscr;
  writeln(' ');
  writeln(' STATUS UPDATE ');
  writeln(' ');
  write(' Request ID: ');
  readln(EnteredRequestId);
  writeln(' ');
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
      read(openreqfile,newreq);
      If newreq.reqId=EnteredRequestId Then
        Begin
          writeln(' You are about to update
',newreq.reqUserForename,' ',newreq.reqUserSurname,
                  ''s request.');
          writeln(' ');
          Repeat
            If newreq.reqAssignedTech = ' ' Then
              Begin
                writeln(' No technician has been assigned to this
request.');
                writeln(' Before continuing, you must assign a
technician.');
                writeln(' ');
                write(' Enter the assigned technician''s initials:
');
                read(newreq.reqAssignedTech);
                writeln(' ');
              End;
          Until newreq.reqAssignedTech <> ' ';
          writeln(' This request is: ');
          writeln('  1 - Open');
          writeln('  2 - Being investigated');
          writeln('  3 - Awaiting delivery of parts');
          writeln('  4 - Being resolved');
```

Linear search.

```
          writeln('  5 - Requires ',newreq.reqUserForename,'
',newreq.reqUserSurname,'''s attention'
          );
          writeln('  6 - Ready for collection');
          writeln('  7 - Other');
          write(' Option: ');
          readln(requestStatusUpdateOption);
          writeln(' ');
          writeln(' Would you like to add notes to this request?');
          writeln(' Yes [Y] or No [N]');
          write(' ');
          readln(addNotesOption);
          If addNotesOption In ['n','N'] Then
            Begin
              newreq.reqCurrentStatus := requestStatusUpdateOption;
              clrscr;
              writeln(' ');
              writeln(' THANK YOU ');
              writeln(' ');
              writeln(' The status of this request has been updated
successfully!');
              writeln(' ');
              write(' The new status is: ');
              showstatus(newreq.reqCurrentStatus);
              If newreq.reqAssignedTech <> ' ' Then
                writeln(' This request has been assigned to
',newreq.reqAssignedTech);
              writeln(' ');
       {aaa}
              writeln(' Hit [ENTER] to return to the requests
menu.');
              seek(openreqfile,filepos(openreqfile)-1);
              write(openreqfile,newreq);
              readln;
            End
          Else
            Begin
              writeln(' ');
              writeln(' Please note: you''re limited to 255
characters! ');
              writeln(' ');
              writeln(' Enter your notes below, then hit [ENTER] to
save');
              readln(newReq.reqNotes);
              writeln(' Thanks: your notes have been saved!');
              writeln(' Hit [ENTER] to quit');
              seek(openreqfile,filepos(openreqfile)-1);
              write(openreqfile,newreq);
              writeln(' Hit [ENTER] to return to the requests
menu.');
              readln;
            End;
        End;
    End;
End;
```

```
{ The following procedure allows users to find request information
}
{ by giving the request's ID. Various information about the request
}
{ is then printed up for the user.
}

Procedure FindRequestById;

Var
  userId : integer;                    { id entered by user }
  searchPosition : integer;            { start position within file }
  foundRecord : boolean;               { yes/no: has record been found?
}
  SubMenuOption : integer;             { stores submenu answer }
  x : integer;
Begin
  clrscr;
  searchPosition := 0;
  foundRecord := false;
  writeln(' ');
  writeln('SEARCH BY REQUEST ID');
  writeln(' ');
  write('Enter Request ID:     ');
  readln(UserId);
  writeln(' ');
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
      read(openreqfile,newreq);
      If newreq.reqId=userId Then
        Begin
          writeln('SEARCH RESULTS');
          writeln(' ');
          write('Request ID:          ');
          write(newreq.reqId);
          writeln(' ');
          write('Assigned to:         ');
          write(newreq.reqAssignedTech);
          writeln(' ');
          write('Submission Date:     ');

write(newreq.reqDateDay,'/',newreq.reqDateMonth,'/',newreq.reqDateYe
ar);
          writeln(' ');
          write('Forename:            ');
          write(newreq.reqUserForename);
          writeln(' ');
          write('Surname:             ');
          write(newreq.reqUserSurname);
          writeln(' ');
          write('Department:          ');
          write(newreq.reqUserDept);
          writeln(' ');
          write('System Name:         ');
```

```
            write(newreq.reqSystemName);
            writeln(' ');
            write('System Location:   ');
            write(newreq.reqSystemLoc);
            writeln(' ');
            write('Error Category:    ');
            write(newreq.reqErrorCat);
            writeln(' ');
            write('Error Description: ');
            write(newreq.reqErrorDesc);
            writeln(' ');

{     writeln('** OPTIONS **');
     writeln('1. Edit');
     writeln('2. Delete');
     writeln('3. Quit');
     writeln('Option: ');
     readln(SubMenuOption);
     if SubMenuOption = 1 then EditRequest
     if SubMenuOption = 2 then DeleteRequest
     if SubMenuOption = 3 }
         End;
     End;
  close(openreqfile);
  readln;
End;

{ The following procedure allows users to find request information
}
{ by giving the surname associated with the user who originally
}
{ reported the request.
}

Procedure FindRequestBySurname;

Var
  userSurname : string;          { stores surname entered by user }
  searchPosition : integer;      { start record within file }
  foundRecord : boolean;         { yes/no: record has been found }
  SubMenuOption : integer;       { stores submenu option answer }
  x : integer;
Begin
  clrscr;
  searchPosition := 0;
  foundRecord := false;
  writeln(' ');
  writeln('SEARCH BY SURNAME');
  writeln(' ');
  write('Enter surname:     ');
  readln(UserSurname);

{ for x:=1 to length(newreq.reqUserSurname) do
newreq.reqUserSurname[x]:=upcase(newreq.reqUserSurname[x]); }
  writeln(' ');
  writeln(newreq.reqUserSurname);
```

```
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
      read(openreqfile,newreq);
      If newreq.reqUserSurname=userSurname Then
        Begin
          writeln('SEARCH RESULTS');
          writeln(' ');
          write('Request ID:        ');
          write(newreq.reqId);
          writeln(' ');
          write('Assigned to:       ');
          write(newreq.reqassignedTech);
          writeln(' ');
          write('Forename:          ');
          write(newreq.reqUserForename);
          writeln(' ');
          write('Surname:           ');
          write(newreq.reqUserSurname);
          writeln(' ');
          write('Department:        ');
          write(newreq.reqUserDept);
          writeln(' ');
          write('System Name:       ');
          write(newreq.reqSystemName);
          writeln('  ');
          write('System Location:   ');
          write(newreq.reqSystemLoc);
          writeln(' ');
          write('Error Category:    ');
          write(newreq.reqErrorCat);
          writeln(' ');
          write('Error Description: ');
          write(newreq.reqErrorDesc);
          writeln(' ');

{     writeln('** OPTIONS **');
    writeln('1. Edit');
    writeln('2. Delete');
    writeln('3. Quit');
    writeln('Option: ');
    readln(SubMenuOption);
    if SubMenuOption = 1 then EditRequest
    if SubMenuOption = 2 then DeleteRequest
    if SubMenuOption = 3 }
          If newreq.reqUserSurname<>userSurname Then writeln(
                                                          'This
Request ID does not seem to exist.'
            );
          writeln(' ');
          writeln('To return to main menu, hit [ENTER].');
          readln;
        End;
    End;
  close(openreqfile);
```

```
End;

{ The following procedure acts as a menu to call the FindRequestById
}
{ and FindRequestBySurname procedures. The menu works by accepting
}
{ numeric input from the user.
}

Procedure FindRequest;

Var
  menuOption : integer; { stores answer to navigation prompt }
Begin
  clrscr;
  writeln(' ');
  writeln(' FIND REQUEST ');
  writeln(' ');
  writeln(' Find request by...');
  writeln('  1) ID');
  writeln('  2) Surname');
  writeln(' ');
  writeln('Option: ');
  readln(menuOption);
  If menuOption = 1 Then FindRequestById
  Else If menuOption = 2 Then FindRequestBySurname
  Else writeln('Please enter a valid option!');
End;

{ The following procedure will find an employee based }
{ upon the employee's ID. Each employee is given a    }
{ unique ID when their details are entered into the   }
{ system.                                             }

Procedure FindEmployeeById;

Var
  userId : integer;                { id entered by user }
  searchPosition : integer;        { start position in file }
  foundRecord : boolean;           { yes/no: record has been found }
  SubMenuOption : integer;         { answer to navigation option }
  x : integer;
Begin
  clrscr;
  searchPosition := 0;
  foundRecord := false;
  writeln(' ');
  writeln('SEARCH BY EMPLOYEE ID');
  writeln(' ');
  write('Enter Employee ID:     ');
  readln(UserId);
  { for x:=1 to length(newreq.empId) do
newreq.empId[x]:=upcase(newreq.empId[x]); }
  writeln(' ');
 {writeln(newreq.reqId);}
  assign(employeesfile,employeefilename);
```

```
  reset(employeesfile);
  While Not eof(employeesfile) Do
    Begin
      read(employeesfile,newemp);
      If newemp.empId=userId Then
        Begin
          writeln('SEARCH RESULTS');
          writeln(' ');
          write('Employee ID:        ');
          write(newreq.reqId);
          writeln(' ');
          write('Forename:          ');
          write(newreq.reqUserForename);
          writeln(' ');
          write('Surname:           ');
          write(newreq.reqUserSurname);
          writeln(' ');
          write('Department:        ');
          write(newreq.reqUserDept);
          writeln(' ');
          write('System Name:       ');
          write(newreq.reqSystemName);
          writeln(' ');
          write('System Location:   ');
          write(newreq.reqSystemLoc);
          writeln(' ');
          write('Error Category:    ');
          write(newreq.reqErrorCat);
          writeln(' ');
          write('Error Description: ');
          write(newreq.reqErrorDesc);
          writeln(' ');

{     writeln('** OPTIONS **');
    writeln('1. Edit');
    writeln('2. Delete');
    writeln('3. Quit');
    writeln('Option: ');
    readln(SubMenuOption);
    if SubMenuOption = 1 then EditRequest
    if SubMenuOption = 2 then DeleteRequest
    if SubMenuOption = 3 }
        End;
    End;
  close(openreqfile);
End;

Procedure FindAssetById;

Var assetId : integer;          { asset id entered by user }
Begin
  clrscr;
  writeln('Enter asset ID: ');
  readln(assetId);
  assign(assetsfile,assetsfilename);
  reset(assetsfile);
```

```
  While Not eof(assetsfile) Do
    Begin
      read(assetsfile,newast);
      If newast.astId=assetId Then
        Begin
          writeln(' ');
          writeln(' SEARCH RESULTS');
          writeln(' ');
          writeln(' Asset ID: ',newast.astId);
          writeln(' Mfgr:     ',newast.astMfgr);
          writeln(' Model:    ',newast.astModel);
          writeln(' Location: ',newast.astLoc);
          writeln(' Pur Date: ',newast.astPurchDate);
          writeln(' Retailer: ',newast.astPurchRetl);
          writeln(' Warranty: ',newast.astWarrExpir);
          writeln(' ');
          writeln(' Hit [ENTER] to quit.');
          readln;
        End;
      close(assetsfile);
    End;
End;

Procedure FindAssetByType;

Var assetType : string;                { asset type entered by user }
Begin
  clrscr;
  writeln(' ');
  writeln(' Types are declared as: ');
  writeln('  1 = Workstation');
  writeln('  2 = Server');
  writeln('  3 = Whiteboard');
  writeln('  4 = Projector');
  writeln('  5 = Peripheral');
  writeln('  6 = Other');
  writeln(' ');
  writeln('Enter type number: ');
  readln(assetType);
  If assetType>=7 Then
    Begin
      writeln(' Please enter a valid type number!');
    End
    assign(assetsfile,assetsfilename);
  reset(assetsfile);
  While Not eof(assetsfile) Do
    Begin
      read(assetsfile,newast);
      If newast.astType=assetType Then
        Begin
          writeln(' ');
          writeln(' SEARCH RESULTS');
          writeln(' ');
          writeln(' Asset ID: ',newast.astId);
          writeln(' Mfgr:     ',newast.astMfgr);
          writeln(' Model:    ',newast.astModel);
```

```
          writeln(' Location: ',newast.astLoc);
          writeln(' Pur Date: ',newast.astPurchDate);
          writeln(' Retailer: ',newast.astPurchRetl);
          writeln(' Warranty: ',newast.astWarrExpir);
          writeln(' ');
          writeln(' Hit [ENTER] to quit.');
          readln;
        End;
      close(assetsfile);
    End;
End;

{ The following procedure checks to find an asset's details }
{ when the user provides the manufacturer's name.           }

Procedure FindAssetByManufacturer;

Var assetMfgr : string;          { asset mfgr entered by user }
Begin
  clrscr;
  writeln(' ');
  writeln('Enter manufacturer: ');
  readln(assetMfgr);
  assign(assetsfile,assetsfilename);
  reset(assetsfile);
  While Not eof(assetsfile) Do
    Begin
      read(assetsfile,newast);
      If newast.astMfgr=assetMfgr Then
        Begin
          writeln(' ');
          writeln(' SEARCH RESULTS');
          writeln(' ');
          writeln(' Asset ID: ',newast.astId);
          writeln(' Mfgr:     ',newast.astMfgr);
          writeln(' Model:    ',newast.astModel);
          writeln(' Location: ',newast.astLoc);
          writeln(' Pur Date: ',newast.astPurchDate);
          writeln(' Retailer: ',newast.astPurchRetl);
          writeln(' Warranty: ',newast.astWarrExpir);
          writeln(' ');
          writeln(' Hit [ENTER] to quit.');
          readln;
        End;
      close(assetsfile);
    End;
End;

{ The following procedure checks to find the asset's details }
{ when given the asset's location by the user. Multiple      }
{ assets may appear when searching by location.              }

Procedure FindAssetByLocation;

Var assetLoc : string;           { asset location entered by user }
Begin
```

```
  clrscr;
  writeln(' ');
  writeln('Enter asset location: ');
  readln(assetLoc);
  assign(assetsfile,assetsfilename);
  reset(assetsfile);
  While Not eof(assetsfile) Do
    Begin
      read(assetsfile,newast);
      If newast.astLoc=assetLoc Then
        Begin
          writeln(' ');
          writeln(' SEARCH RESULTS');
          writeln(' ');
          writeln(' Asset ID: ',newast.astId);
          writeln(' Mfgr:     ',newast.astMfgr);
          writeln(' Model:    ',newast.astModel);
          writeln(' Location: ',newast.astLoc);
          writeln(' Pur Date: ',newast.astPurchDate);
          writeln(' Retailer: ',newast.astPurchRetl);
          writeln(' Warranty: ',newast.astWarrExpir);
          writeln(' ');
          writeln(' Hit [ENTER] to quit.');
          readln;
        End;
      close(assetsfile);
    End;
End;

{ This procedure writes up the Assets submenu, and identifies }
{ correct menu options which can be called from within it to  }
{ access different features.                                  }

Procedure FindAsset;

Var menuOption : integer;     { navigation option }
Begin
  clrscr;
  writeln('Find asset by...');
  writeln('1. Asset ID');
  writeln('2. Asset Type');
  writeln('3. Asset Manufacturer');
  writeln('4. Asset Location');
  writeln('Option: ');
  readln(menuOption);
  If menuOption = 1 Then FindAssetById
  Else If menuOption = 2 Then FindAssetByType
  Else If menuOption = 3 Then FindAssetByManufacturer
  Else If menuOption = 4 Then FindAssetByLocation
  Else writeln('Please enter a valid option!');
End;

{ The following procedure checks to make sure that the files }
{ require for the system to store data correctly exist.       }
{ If they don't, the procedure creates the files. The file   }
{ locations can be declared at the top of the program source }
```

Quite a lot of very similar search procedures.

```
Procedure MakeFiles;

Var zero: integer;              { sets start id to zero }
Begin
  assign(openreqfile,openreqfilename);
  rewrite(openreqfile);
  close(openreqfile);
{---}
  assign(employeesfile,employeefilename);
  rewrite(employeesfile);
  close(employeesfile);
{---}
  assign(assetsfile,assetsfilename);
  rewrite(assetsfile);
  close(assetsfile);
{---}
  zero := 0;
  assign(idfile,idfilename);
  rewrite(idfile);
  write(idfile,zero);
  write(idfile,zero);
  close(idfile);
End;


{ procedure AddNewEmployee;
 var newemp : emptype;
begin
 clrscr;
 writeln(' ');
 writeln('       ADD EMPLOYEE ');
 writeln(' ');
 write('ID:          ');
 readln(newemp.empId);
 write('Forename:    ');
 readln(newemp.empForename);
 write('Surname:     ');
 readln(newemp.empSurname);
 write('Dept.:       ');
 readln(newemp.empDept);
 write('Phone:       ');
 readln(newemp.empPhone);
 assign(employeesfile,employeefilename);
 reset(employeesfile);
 seek(employeesfile,filesize(employeesfile));
 write(employeesfile,newemp);
 close(employeesfile);
 clrscr;
 writeln(' ');
 writeln(' THANK YOU! ');
 writeln(' ');
 writeln(' Many thanks for adding ',newEmp.empForename,'
',newEmp.empSurname,' to the system.');
 writeln(' Their details have been accepted and saved.');
 writeln(' To return to the main menu, hit [ENTER].');
```

```
 readln;
end;        }

{ The following procedure adds a new asset to the system. The system
}
{ can then be used to recall details regarding the asset, if - for
}
{ example - it needs to be serviced.
}

Procedure AddNewAsset;

Var newast : asttype;              { asset structure definition }
  optionchoice : char;           { submenu navigation }
Begin
  clrscr;
  writeln(' ');
  writeln('     ADD NEW ASSET      ');
  writeln(' ');
  writeln(' Asset types include: ');
  writeln('  1 = Workstation');
  writeln('  2 = Server');
  writeln('  3 = Whiteboard');
  writeln('  4 = Projector');
  writeln('  5 = Peripheral');
  writeln('  6 = Other');
  writeln(' ');
  write(' Asset Type:                ');
  readln(newast.astType);
  write(' Manufacturer:              ');
  readln(newast.astMfgr);
  write(' Model:                     ');
  readln(newast.astModel);
  write(' Location:                  ');
  readln(newast.astLoc);
  If length(newast.astLoc)<2 Then
    Begin
      writeln(' ');
      writeln(' An error has been detected.');
      writeln(' Are you sure this is a real room location?');
      writeln(' Hit C to continue... ');
      writeln(' ');
      readln(optionchoice);
      If (optionchoice) = 'C' Then
        Begin
          clrscr;
          writeln('     ADD NEW ASSET      ');
          writeln(' ');
          write(' Asset Type:              ',newast.astType);
          write(' Manufacturer:            ',newast.astMfgr);
          write(' Model:                   ',newast.astModel);
          write(' Location:                ',newast.astLoc);
          write(' Purchase Date:           ',newast.astPurchDate);
          write(' Purchase Retailer:       ',newast.astPurchRetl);
          write(' Warranty Expires:        ',newast.astWarrExpir);
          assign(assetsfile,assetsfilename);
```

```
          reset(assetsfile);
          seek(assetsfile,filesize(assetsfile));
          latestreqID := latestreqID+1;
          newreq.reqID := latestreqID;
          write(assetsfile,newast);
          close(assetsfile);
        End;
      clrscr;
      writeln(' ');
      writeln(' THANK YOU ');
      writeln(' ');
      writeln(' The asset details have been accepted succesfully.
');
      writeln(' A copy of the details accepted are shown below.');
      writeln(' Please attach the Asset ID to the asset itself.');
      writeln(' ');
      writeln(' Asset ID:          ',newast.astId);
      writeln(' Type:              ',newast.astType);
      writeln(' Manufacturer:      ',newast.astMfgr);
      writeln(' Model:             ',newast.astModel);
      writeln(' Location:          ',newast.astLoc);
      writeln(' Purchase Date:     ',newast.astPurchDate);
      writeln(' Purchase Retailer: ',newast.astPurchRetl);
      writeln(' Warranty Expires:  ',newast.astWarrExpir);
      writeln(' ');
      writeln(' To return to main menu, hit [ENTER].');
      readln;
    End;
End;

{ The following procedure allows a user to delete a request's
details. }
{ It requires the use of the system-wide password.
}

Procedure DeleteRequest;

Var RequestID : integer;    { request id entered by user }
Begin
  clrscr;
  checkpassword;
  clrscr;
  writeln(' ');
  writeln(' DELETE REQUEST');
  writeln(' ');
  writeln(' Request ID: ');
  read(RequestID);
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
      read(openreqfile,newreq);
      If newreq.reqId=RequestID Then
        Begin
          seek(openreqfile,filepos(openreqfile)-1);
          newreq.reqId := 0;
```

```
          write(openreqfile,newreq);
          writeln(' Request deleted');
          readln;
        End
      Else writeln(' This request ID cannot be found.');
      readln;
    End;
  close(openreqfile);
End;

{ The following procedure writes the requests menu, and associates
certain }
{ numerical options with it, allowing the user to enter deeper
features    }
{ associated with requests.
}

Procedure writeRequestsMenu;

Var
  menuOption : integer;    { submenu navigation option }
Begin
  Repeat
    clrscr;
    writeln(' ');
    writeln('    REQUESTS    ');
    writeln(' ');
    writeln(' -----------------');
    writeln(' #   ACTION');
    writeln(' -----------------');
    writeln(' 1   ADD REQUEST');
    writeln(' 2   VIEW REQUEST');
    writeln(' 3   UPDATE REQUEST');
    writeln(' 4   DELETE REQUEST');
    writeln(' 5   RETURN TO MAIN MENU');
    writeln(' -----------------');
    writeln(' OPTION: ');
    readln(menuOption);
    If menuOption=1 Then AddNewRequest;
    If menuOption=2 Then FindRequest;
    If menuOption=3 Then UpdateRequestStatus;
    If menuOption=4 Then DeleteRequest;
  Until menuOption = 5;
End;

{ The following procedure allows a user to delete an asset as long
as they }
{ know the system-wide password. The password is enforced here so
that     }
{ data is kept as secure and integral as possible, without limiting
use to }
{ a considerable extent.
}

Procedure DeleteAsset;
```

```
Var assetID : integer;     { asset id entered by user }
Begin
  clrscr;
  checkpassword;
  clrscr;
  writeln(' ');
  writeln(' Delete Asset');
  writeln(' ');
  writeln(' Asset ID: ');
  read(assetID);
  assign(assetsfile,assetsfilename);
  reset(assetsfile);
  While Not eof(assetsfile) Do
    Begin
      read(assetsfile,newast);
      If newast.astId=assetID Then
        Begin
          seek(assetsfile,filepos(assetsfile)-1);
          newast.astId := 0;
          newast.astType := ' ';
          newast.astMfgr := ' ';
          newast.astModel := ' ';
          newast.astLoc := ' ';
          newast.astPurchDate := ' ';
          newast.astPurchRetl := ' ';
          newast.astWarrExpir := ' ';
          write(assetsfile,newast);
          writeln(' Asset deleted!');
          readln;
        End
      Else writeln(' This asset cannot be found!');
      readln;
    End;
  close(assetsfile);
End;

{ The following procedures writes the Assets menu, and defines it's
associated }
{ numerical options, used to access features deeper within the
Assets fields.  }

Procedure writeAssetsMenu;

Var menuOption : integer;  { submenu navigation option }
Begin
  Repeat
    clrscr;
    writeln(' ');
    writeln('     ASSETS    ');
    writeln(' ');
    writeln(' -----------------');
    writeln(' #   ACTION');
    writeln(' -----------------');
    writeln(' 1   ADD NEW ASSET');
    writeln(' 2   FIND ASSET');
    writeln(' 3   DELETE ASSET');
```

```
    writeln(' 4   RETURN TO MAIN MENU');
    writeln(' -----------------');
    readln(menuOption);
    If menuOption=1 Then AddNewAsset;
    If menuOption=2 Then FindAsset;
    If menuOption=3 Then DeleteAsset;
  Until menuOption = 4;
End;


{procedure writeEmployeesMenu;
var
 menuOption : integer;
begin
 repeat
 clrscr;
 writeln(' ');
 writeln('     EMPLOYEES    ');
 writeln(' ');
 writeln(' -----------------');
 writeln(' #   ACTION');
 writeln(' -----------------');
 writeln(' 1   ADD NEW EMPLOYEE');
 writeln(' 2   FIND EMPLOYEE');
 writeln(' 3   DELETE EMPLOYEE');
 writeln(' 4   RETURN TO MAIN MENU');
 writeln(' -----------------');
 readln(menuOption);
 if menuOption=1 then AddNewEmployee;
 if menuOption=2 then FindEmployee;
 if menuOption=3 then {DeleteEmployee;
 until menuOption=4;
 end;                                }

{ The following procedure allows the user to set the system-wide
password. }

Procedure SetPassword;

Var
  useroldpassword   : string; { Stores old password according to
user }
  oldpassword       : string; { Stores old password according to
system }
  newpassword       : string; { Stores new password according to
user }
  newpasswordverify : string; { Stores verify password. Used to
check. }
Begin
  If length(oldpassword)>1 Then
    Begin
      writeln(' Enter existing password: ');
      readln(useroldpassword);
      If useroldpassword=oldpassword Then
        Begin
          writeln(' Enter new password: ');
```

```
          readln(newpassword);
          writeln(' Confirm new password: ');
          readln(newpasswordverify);
          If newpassword=newpasswordverify Then
            oldpassword := newpassword;
        End;
    End
  Else
    Begin
      clrscr;
      writeln(' ');
      writeln(' PASSWORD');
      writeln(' ');
      writeln(' You need to set a password!');
      write(' Enter a password: ');
      readln(newpassword);
      write(' Confirm password: ');
      readln(newpasswordverify);
      If newpasswordverify=newpassword Then
        Begin
          globalpassword := newpassword;
          textcolor(green);
          writeln(' ');
          writeln(' SUCCESS!');
          textcolor(lightgray);
          writeln(' Your password has been set accordingly.');
          writeln(' ');
          writeln(' Hit [ENTER] to quit to main menu.');
          readln;
        End
      Else
        Begin
          textcolor(red);
          writeln(' ');
          writeln(' ERROR!');
          textcolor(lightgray);
          writeln(' An error was detected in the passwords
entered.');
          writeln(' Please try again.');
          writeln(' ');
          writeln(' Hit [ENTER] to quit to main menu.');
        End;
      readln;
    End;
End;

{ Asks the user if they are sure they wish to quit the program. }

Function areYouSureQuit : boolean;

Var QuitAnswer : char;          { answer to "confirm quit" question }
  quit       : boolean;       { forces program to quit when = true }
Begin
  Repeat
    clrscr;
    writeln(' ');
```

```
    writeln(' Checking for amended data...');
    delay(255);
    delay(255);
    writeln(' Please wait: saving any amended data...');
    delay(255);
    delay(255);
    delay(255);
    clrscr;
    writeln(' ');
    writeln(' DATA SAVED SUCCESSFULLY!');
    writeln(' It is now safe to quit this program.');
    writeln(' ');
    writeln(' Are you sure you wish to quit?');
    writeln(' Type Y for YES or N for NO, then hit ENTER to
confirm.');
    write(' Answer: ');
    read(QuitAnswer);
    QuitAnswer := upcase(QuitAnswer);
  Until (QuitAnswer = 'Y') Or (QuitAnswer = 'N');
  If QuitAnswer = 'Y' Then quit := true
  Else quit := false;
  areYouSureQuit := quit;
End;

{ Prints up the global settings menu, allowing users to change  }
{ the password or delete all data.                              }

Procedure GlobalSettings;

Var menuoption : integer; { submenu navigation option }
  areyousure : char;    { are you sure - y = do, n = don't }
Begin
  ClrScr;
  writeln(' ');
  writeln('    SETTINGS   ');
  writeln(' ');
  writeln(' ------------------');
  writeln(' #   ACTION');
  writeln(' ------------------');
  writeln(' 1   CLEAR DATA');
  writeln(' 2   SET PASSWORD');
  writeln(' 4   RETURN TO MAIN MENU');
  writeln(' ------------------');
  writeln(' OPTION: ');
  readln(menuOption);
  If menuoption = 1 Then
    Begin
      checkpassword;
      clrscr;
      writeln(' ');
      textcolor(red);
      writeln('            --- WARNING ---');
      textcolor(lightgray);
      writeln(' ');
      writeln(' Are you sure you want to clear all data?');
      writeln(' Cleared data is irretrievable.');
```

```
      writeln(' ');
      writeln(' Hit Y for YES or N for NO: ');
      writeln(' ');
      write(' ');
      read(areyousure);
      clrscr;
      If areyousure = 'Y' Then
        clrscr;
      writeln(' ');
      writeln(' Clearing saved data...');
      delay(255);
      delay(255);
      MakeFiles;
      writeln(' ');
      writeln(' All data files have been cleared!');
      writeln(' Hit [ENTER] to continue.');
      readln;
      readln;
      If areyousure = 'N' Then
        Begin
          writeln('Data has not been cleared.');
          writeln('All data has been preserved.');
          GlobalSettings;
        End;
    End;
  If menuoption = 2 Then SetPassword;
End;

{ The following procedure allows the user to view details associated
with an }
{ asset.
}

Procedure viewAsset;

Var
  enteredAssetId : integer;  { asset id entered by user }
Begin
  assign(assetsfile,assetsfilename);
  reset(assetsfile);
  writeln(' VIEW ASSET');
  writeln(' ');
  writeln(' Enter Asset ID: ');
  readln(enteredAssetId);
  If newast.astId=enteredAssetId Then
    Begin
      close(assetsfile);
    End;
End;

{ The following procedure allows the user to track the status of a
request }
{ as it progresses through the system. It relies heavily on other
files    }
{ and procedures to achieve it's aims.
}
```

```
Procedure viewTracking;

Var TrackingRef : integer;      { tracking ref entered by user }
  searchPosition : integer;   { start position in file }
  foundRecord : boolean;      { yes/no: record found }
  subMenuOption : integer;    { submenu navigation option }
Begin
  clrscr;
  writeln(' ');
  writeln(' TRACKING   ');
  writeln(' ');
  writeln(' This service allows you to check the status of any open
or archived requests.');
  writeln(' You can view: ');
  writeln('   - Request details');
  writeln('   - Historical status updates');
  writeln('   - Engineer notes');
  writeln(' ');
  textcolor(yellow);
  writeln(' A tracking reference is required to use this service.');
  textcolor(lightgray);
  writeln(' Tracking references are available once you have
submitted a request.');
  write(' It is');
  textcolor(red);
  write(' NOT ');
  textcolor(lightgray);
  write('possible to relocate a tracking reference if misplaced.');
  writeln(' ');
  writeln(' ');
  writeln(' Tracking Reference: ');
  writeln(' ');
  writeln(' To quit, press 0 on the numpad, then hit [ENTER].');
  gotoxy(23,14);
  searchPosition := 0;
  foundRecord := false;
  readln(TrackingRef);
  writeln(' ');
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
     read(openreqfile,newreq);
      If newreq.reqId=TrackingRef Then
       Begin
         foundrecord := true;
         clrscr;
         writeln(' ');
         writeln(' TRACKING');
         writeln(' ');
         writeln(' Tracking options for request ',TrackingRef);
         writeln(' ');
         writeln(' #  ACTION');
         writeln(' ------------------------------------');
         writeln(' 1  Display notes and status         ');
```

```
            writeln(' 2  Display original request details ');
            writeln(' ------------------------------------');
            write(' Opt: ');
            read(subMenuOption);
            If subMenuOption = 2 Then FindRequestById
            Else
              Begin
                clrscr;
                writeln(' ');
                writeln(' TRACKING');
                writeln(' ');
                writeln(' Tracking Ref: ');
                write(' ',newreq.reqId);
                writeln(' ');
                writeln(' ');
                writeln(' Current Status: ');
                showstatus(newreq.reqCurrentStatus);
                writeln(' ');
                writeln(' Request Notes: ');
                write(' ',newreq.reqNotes);
                writeln(' ');
                writeln(' ');
                writeln(' Assigned to: ');
                writeln(' ',newreq.reqAssignedTech);
              {ppp}
                writeln(' ');
                writeln(' Press [ENTER] to return to main menu');
                readln;
            {bbb}
              End;
          End;
      End;
    If foundrecord=false Then
      Begin
        clrscr;
        writeln(' ');
        writeln(' TRACKING ');
        writeln(' ');
        textcolor(red);
        write(' ERROR: ');
        textcolor(lightgray);
        write('No tracking reference has been found!');
        writeln(' ');
        writeln(' Are you sure this tracking reference exists?');
        writeln(' ');
        write(' Hit [ENTER] to continue.');
      End;
    close(openreqfile);
    readln;
End;

{ Provides a way for a technician to get a personalised output of
jobs }
{ assigned to them which need to be completed.
}.
```

```
Procedure agenda;

Var enteredInitials : string[5];            { initials entered by
user }
  currentjobs: array[1..20] Of reqtype;    { defines current jobs
}
  jcount: integer;                          { job count }
  j : integer;
  x : integer;                             { gotoxy location - vert }
  y : integer;                             { gotoxy location - horz }
  temp: reqtype;                            { used in sorting jobs by
status }
  a,b: integer;                             { used in job status
lookup }
Begin
  x := 0;
  clrscr;
  writeln(' ');
  writeln(' AGENDA');
  writeln(' ');
  write(' Enter initials: ');
  read(enteredInitials);
  jcount := 0; { sets value of jcount to 0 }
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do { as long as not end of file, do: }
    Begin
      read(openreqfile,newreq);

WRITELN('*',newreq.reqAssignedTech,'*',newreq.reqCurrentStatus,'
',newreq.reqID);
   { writes assigned tech, current status, id - separated by
asterisks }
      readln;
      If (enteredinitials=newreq.reqAssignedTech) And
         (newreq.reqCurrentStatus<>7) And (newreq.reqID<>0)
       {if enteredinitials equals assigned tech for a request AND }
       {if newreq.reqCurrentStatus does not equal 7 AND }
       {if newreq.reqID does not equal 0 - ie deleted, then: }
        Then
        Begin
          jcount := jcount+1;
        { add 1 to value of jcount }
          currentjobs[jcount] := newreq;
       { set value of currentjobs 1 thru 7 to newreq values }
        End;
    End;
  close(openreqfile);
  readln;

  If jcount>0 Then
    Begin

      gotoxy(1,2);
      y := 4;
      gotoxy(1,y);
```

Counters in array for different types of job.

```
      clrscr;

      For a:=1 To jcount-1 Do
        For b:=1 To jcount-1 Do
          If
currentjobs[b].reqcurrentstatus>currentjobs[b+1].reqcurrentstatus
Then
     { if value of current job's status is more than next job's
status, then: }
            Begin
              temp := currentjobs[b]; { set temp to currentjobs
value }
              currentjobs[b] := currentjobs[b+1]; { store next value
in currentjobs }
              currentjobs[b+1] := temp; { store next value in temp }
            End;


      For j:=1 To jcount Do
        Begin
          gotoxy(2,2); { go to vert: 2, horz: 2 - pattern similar
throughout proc }
          writeln('#');
          gotoxy(1,y);
          writeln(' ',currentjobs[j].reqID);
          gotoxy(6,2);
          writeln('DATE');
          gotoxy(5,y);
          writeln('
',currentjobs[j].reqDateDay,'/',newreq.reqDateMonth,'/',newreq.reqDa
teYear Mod
                  100);
    { write date in dd/mm/yy mod 100 to achieve readable format }
          gotoxy(14,2);
          writeln('S');
          gotoxy(13,y);
          writeln(' ',currentjobs[j].reqCurrentStatus);
          gotoxy(16,2);
          writeln('NAME');
          gotoxy(15,y);
          writeln(' ',currentjobs[j].reqUserForename[1],'
',currentjobs[j].reqUserSurname);
          gotoxy(28,2);
          writeln('DEPT');
          gotoxy(27,y);
          writeln(' ',currentjobs[j].reqUserDept);
          gotoxy(35,2);
          writeln('LOCATION');
          gotoxy(34,y);
          writeln(' ',currentjobs[j].reqSystemLoc,'
','(',currentjobs[j].reqSystemName,')');
          gotoxy(48,2);
          writeln('CATEGORY');
          gotoxy(47,y);
          writeln(' ',currentjobs[j].reqErrorCat);
          gotoxy(58,2);
```

```
          writeln('DESCRIPTION');
          gotoxy(57,y);
          writeln(' ',currentjobs[j].reqErrorDesc);
{    gotoxy(61,y);
    writeln('i',newreq.reqNotes);}
{    gotoxy(60,y);  }
          y := y+1;
        End;

    End
  Else
    writeln(' ');
  writeln(' ');
  writeln(' Hit [ENTER] to return to main menu.');
  readln;
End;

{ Counts the number of jobs in the system, and places them in
categories. }
{ This is used by the GenerateStatistics procedure.
}

Procedure countJobs;

Var j: integer;       { current record }
  alreadyexists : boolean;  { yes/no: this job already exists }

Begin
  For j:=1 To 10 Do
    Begin
      jobcountbytech[j].jobcount := 0;   { set all values of
technicians job counts to 0 }
      jobcountbytech[j].assignedtech := '';  { set assigned tech
value to blank }
    End;
  techcount := 0;
  GlobalTotalJobsCount := 0;  { set GlobalTotalJobsCount to 0:
globally declared }
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  While Not eof(openreqfile) Do
    Begin
      read(openreqfile,newreq);
      If newreq.reqID>0 Then { if request ID = more than 0, ie not
deleted then: }
        Begin
          GlobalTotalJobsCount := GlobalTotalJobsCount+1;
        { add 1 to GlobalTotalJobsCount for every job in file }
        {writeln('Read job with status: ',newreq.reqCurrentStatus);}
          readln;
          JobCountByStatus[newreq.reqCurrentStatus] :=
JobCountByStatus[newreq.reqCurrentStatus]+1;
        { jump to next job in line }
          If techcount=0 Then { if technician count = 0 then }
            Begin
              jobcountbytech[1].jobcount := 1;
```

```
            { set technician count to 1 }
              jobcountbytech[1].assignedtech :=
newreq.reqassignedtech;
            { set this assigned tech to global assigned tech }
              techcount := 1;
            End
          Else
            Begin
              alreadyexists := false;
              For j:=1 To techcount Do
                If
jobcountbytech[j].assignedtech=newreq.reqassignedtech Then
                  Begin
                    jobcountbytech[j].jobcount :=
jobcountbytech[j].jobcount+1;
                    alreadyexists := true;
                  End;
              If alreadyexists=false Then
                Begin
                  techcount := techcount+1;
                  jobcountbytech[techcount].jobcount := 1;
                  jobcountbytech[techcount].assignedtech :=
newreq.reqassignedtech;

                End;
            End;
        End;
    End;
  close(openreqfile);
  readln;
End;

{ Allows the user to view statistics about the jobs on the system. }
{ Password protected as some users may find this data to be
sensitive. }

Procedure viewCurrentStatistics;

Var j : integer;
Begin
  checkpassword;   { used to check password: statistics may be
deemed sensitive }
  countjobs; { call count jobs procedure }
  clrscr;
  writeln(' ');
  writeln('    CURRENT STATISTICS');
  writeln(' ');                                    {xxx}
  writeln(' Currently, there are ',GlobalTotalJobsCount,' requests
in the system, of which: ');
  writeln('  - ',JobCountByStatus[1],' jobs are deemed open');
  writeln('  - ',JobCountByStatus[2],' jobs are under
investigation');
  writeln('  - ',JobCountByStatus[3],' jobs are awaitng parts');
  writeln('  - ',JobCountByStatus[4],' jobs are being resolved');
  writeln('  - ',JobCountByStatus[5],' jobs require attention');
```

```
  writeln('  - ',JobCountByStatus[6],' jobs are ready to be
collected');
  writeln('  - ',JobCountByStatus[7],' jobs are deemed as Other');
 {8 = closed}
  writeln(' ');
  For j:=1 To techcount Do
    writeln(' ',jobcountbytech[j].assignedtech,' currently has
',jobcountbytech[j].jobcount,
            ' jobs assigned.');
  writeln(' ');
  assign(openreqfile,openreqfilename);
  reset(openreqfile);
  seek(openreqfile,filepos(openreqfile)-1);
  writeln(' The last request was submitted on:
',newreq.reqDateDay,'/',newreq.reqDateMonth,'/',
          newreq.reqDateYear);
  writeln(' ');
  writeln('                                ');
  readln;
End;

{ Writes up main menu, allowing easy navigation via number keys to
various }
{ submenus, all of which contain features. System designed to be
easy-to-use }
{ and features should be within a short number of options from main
menu. }

Procedure writeMainMenu;

Var
  menuOption : integer; { submenu navigation option }
  quitopt    : boolean; { yes/no: quit system }
Begin
  textcolor(lightgray);  { set text colour to lightgray }
  textbackground(black); { set text colour to black }
{ above formatting options used if colour has not been reset within
procedures }
  Repeat
    clrscr;
    writeln(' ');
    writeln(' MAIN MENU    ');
    writeln(' ');
    writeln(' -----------------');
    writeln(' #   ACTION');
    writeln(' -----------------');
    writeln(' 1   REQUESTS');
    writeln(' 2   ASSETS');
    writeln(' 3   STATISTICS');
    writeln(' 4   AGENDA');
    writeln(' 5   TRACKING');
    writeln(' 6   SETTINGS');
    writeln(' 0   QUIT');
    writeln(' -----------------');
    write(' OPT: ');
    readln(menuOption);
```

```
    If menuOption=1 Then writeRequestsMenu;
    If menuOption=2 Then writeAssetsMenu;
    If menuOption=3 Then viewCurrentStatistics;
    If menuOption=4 Then agenda;
    If menuOption=5 Then viewTracking;
    If menuOption=6 Then GlobalSettings;
    If menuOption=0 Then quitopt := areYouSureQuit;
  Until (menuOption=0) Or (quitopt=true);
End;

{ Main program code follows: yes, it really is that short! }

Begin
  clrscr;           { clears screen }
  loadids;          { loads last-used ids }
  WriteMainMenu;    { writes main menu, allowing access to submenus
+ features }
  clrscr;           { clears screen }
End.   { ends program completely }
```

SYSTEM MAINTENANCE

System Overview

The system is programmed in the Pascal programming language. All data entered into the system, or produced as a result of processing, is stored in a number of files which are available to be accessed by procedures and functions throughout the program. This makes the sharing of data within the system fast, and is fairly secure as the files are only available to be called from within the program. Furthermore, data which is regularly needed to be accessed and edited by procedures and functions, is stored within arrays. The files created by the system do not use excessive disk space, and can therefore be backed up quickly and easily. The system also includes some file management tools from within the Settings menu.

Brief overview.

Brief Summary of Features of Packages Used

My program was written entirely by myself in the Pascal programming language. I have not, therefore, used a package to help develop my program.

Detailed Algorithm Design

Good plans for two algorithms.

Tracking procedure

This procedure tracks the current status of any given job at any point in time. It is implemented in the procedure *viewTracking*, and uses the following variables:

TrackingRef	The tracking reference entered by the user when *viewTracking* is called.
searchPosition	The position in the OpenRequests file from which to begin looking for the request that meets the TrackingRef.
foundRecord	Accepts boolean input, allowing the system to recognise when a record matching TrackingRef has been found. This is used to stop the system from searching once a matching record has been found.
subMenuOption	Used to work out where to send the user once the TrackingRef has been entered. The user can either continue to see current job status, or can be referred to the *FindRequestById* procedure which would display the original request details.
reqID	The ID of the request found
reqCurrentStatus	The current status of the found request
reqNotes	Notes associated with the found request
reqAssignedTech	Initials of technicians assigned to this request

The following pseudocode shows how this procedure is implemented within the system:

```
Start procedure "viewTracking"
Define TrackingRef, searchPosition and subMenuOption variables
as integers
Define foundRecord variable as Boolean
Clear entire screen
Write blank line
Write "tracking" header
Write blank line
Write "This service allows you to check the status of any open
or archived requests"
Write "You can view:"
Write "- Request details"
Write "- Historical status updates"
Write "- Engineer notes"
Write blank line
Change text colour to yellow
Write "A tracking reference is required to use this service."
Change text colour to light grey
Write "Tracking references are available once you have
submitted a request"
Write "It is NOT possible to relocate a tracking reference if
misplaced."
Write blank line
Write blank line
Write "Tracking reference: "
Write blank line
Write "To quit, press 0 on the numpad, then hit [ENTER]."
Go to position 23, 14 on the display
Set value of searchPosition to 0
Set value of foundRecord to false
Read value entered into TrackingRef
Write blank line
Assign open requests file using directories specified at start
of program
Reset open requests file, ready for next use
Whilst not end of open requests file do
    Begin
    Read request details from openreqfile,newreq
    If entered reqId equals value of TrackingRef then
        Begin
        Set foundRecord to true
        Clear screen
        Write blank line
        Write "Tracking options for request ", write
tracking reference
        Write blank line
        Write "#  Action"
        Write blank line
        Write "--------------------"
```

```
          Write "1  Display notes and status"
          Write "2  Display original request details"
          Write "--------------------"
          Write "Opt: "
          Read subMenuOption
          If subMenuOption = 2 then call FindRequestById
procedure
          Else
               Clear screen
               Write blank line
               Write "TRACKING"
               Write blank line
               Write "Tracking Ref: "
               Write request ID
               Write blank line
               Write blank line
               Write "Current Status: "
               Put newreq.reqCurrentStatus into CurrentStatus
procedure
               Write CurrentStatus value
               Write blank line
               Write "Request Notes: "
               Write value of newreq.reqValue
               Write blank line
               Write blank line
               Write "Assigned to: "
               Write value of newreq.reqAssignedTech
               Write blank line
               Write "Press [ENTER] to return to main menu"
               Read line
     End while
If FoundRecord = false, then
     Clear screen
     Write blank line
     Write "TRACKING"
     Write blank line
     Change text colour to red
     Write " ERROR: "
     Change text colour to light grey
     Write "No tracking reference has been found!"
     Write blank line
     Write "Are you sure this tracking reference exists?"
     Write blank line
     Write "Hit [ENTER] to continue"
End If
Close open requests file
Read line
```

Agenda procedure

The agenda procedure is used to display the current jobs which are assigned to a particular technician. It is implemented in the system in the *Agenda* procedure. The variables used are:

EnteredInitials	Initials entered by the user when the procedure is first called.
X	Physical locations on screen, used by gotoxy to determine where to display certain information
Y	
reqAssignedTech	The initials of a technician assigned to a request
reqID	The request's unique ID
reqDateDay	The date on which the request was first reported. The date is separated into three parts, as it is automatically gathered from the system clock.
reqDateMonth	
reqDateYear	
reqCurrentStatus	The current status of the request, in numerical form.
reqUserForename	The full name of the user who originally reported the request.
reqUserSurname	
reqUserDept	The department of the user who originally reported the request.
reqSystemLoc	The location of the system which the request refers to.
reqSystemName	The name of the system which the request refers to.
reqErrorCat	The category of error which the system is reported to have.
reqErrorDesc	A more detailed description of the error.
reqNotes	Miscellaneous notes about this request.

The following pseudocode shows how this procedure is implemented within the system:

```
Start procedure "agenda"
Define enteredInitials variable as string with value limit of
5
Define currentJobs as an array of 1 to 20 of reqType
Define a, b, j, jcount, x and y as integers
Define temp variable of type reqType
Set value of X to 0
Clear screen
Write blank line
Write "AGENDA"
Write blank line
Write "Enter initials: "
Read entered value for previous line into enteredInitials
Set value of jCount to 0
Assign open requests file
Reset open requests file, ready for next use
While not end of open requests file, do
     Open request record
     Write "*", followed by assignedTech, CurrentStatus and
reqId
     Read line
     If:  entered initials equal reqAssignedTech
          reqCurrentStatus = 7
          and newreq.reqID = 0
```

```
     then
          Set value of jCount to jCount+1
          Set newreq to value of currentjobs after jCount
value has been entered
     End If
Close open requests file
Read line

If value of jCount is more than 0, then
     Go to position 1, 2 on screen
     Set value of y to 4
     Go to position 1, y on screen
     Clear screen
     Set value of a to 1
     For a to jcount-1 do
          Set value of b to 1
          For value of b to jcount-1 do
               Place value of b into CurrentJobs.
               If value of reqCurrentStatus is more than the
               value of CurrentJobs + 1 then Set variable temp
               to value of CurrentJobs
                    Add 1 to the value of CurrentJobs
                    Set value of temp to CurrentJobs+1
End If
Set value of j to 1
For j=1 to value of jcount do
          Go to position 2, 2 on screen
          Write "#"
          Go to position 1, y on screen
          Write value of currentJobs.reqID
          Go to position 6, 2 on screen
          Write "DATE" as header
          Go to position 5, y on screen
          Write value of currentJobs.reqDateDay, reqDateMonth,
reqDateYear mod 100
          Go to position 14, 2 on screen
          Write "S" as header
          Go to position 13, y on screen
          Write value of reqCurrentStatus
          Go to position 16, 2 on screen
          Write "NAME" as header
          Go to position 15, y on screen
          Write value of reqUserForename (truncated to 1
character), reqUserSurname
          Go to position 28, 2 on screen
          Write "DEPT" as header
          Go to position 27, y on screen
          Write value of reqUserDept
          Go to position 35, 2 on screen
          Write "LOCATION" as header
          Go to position 34, y on screen
```

```
            Write value of reqSystemLoc, reqSystemName
            Go to position 48, 2 on screen
            Write "CATEGORY" as header
            Go to position 47, y on screen
            Write value of reqErrorCat
            Go to position 58, 2 on screen
            Write "DESCRIPTION" as header
            Go to position 57, y on screen
            Write value of reqErrorDesc
      Y:=Y+1
End For

Else
      Write blank line
      Write blank line
      Write "Hit [ENTER] to return to main menu."
      Read line.
```

Procedure and Variables Lists and Descriptions

Very clear and detailed.

Variable Name	Purpose
OpenReqFile	Stores details about open requests in one file
EmployeesFile	Stores details about employees in one file
AssetsFile	Stores details about assets in one file
IDFile	Stores details about the last used ID in a file
NewReq	Stores all details about one open request
NewEmp	Stores all details about one employee
NewAst	Stores all details about one asset
LatestReqID	Stores the last-used request ID
LatestAstID	Stores the last-used asset ID
GlobalTotalJobsCount	Stores the total number of jobs created

SaveIDs Procedure

This procedure is used to stop the same Request or Asset ID being used more than once, and is called when a new asset or request is saved to file. It is important as it stores the last used request and asset ID in a file, and this information is saved so that it is readily available even after the program has been closed. If this procedure did not exist, the system would begin creating new requests and assets with ID of 0 sequentially every time it was started, resulting in massive loss of data. There are no variables for this procedure

ShowStatus Procedure

Different procedures and functions within the system require the current status of a job to be displayed differently. There are many reasons for this: for example, the Agenda procedure requires the status of a procedure to be displayed in numerical form – from 1 through to 7 – so as to be able to display information about each request neatly on one page. This procedure converts the numerical form of the current status of a request to the alphanumeric form, which is more easily understood by humans. There are no variables for this procedure.

LoadIDs Procedure
This procedure is called whenever a new asset or request is created, and is called before the procedure starts to accept input. It loads the last used IDs, and the procedures calling it can then manipulate the data it outputs. For example, the *addRequest* procedure takes the data ouput and adds a value of 1 to create a sequential and unique ID for a new request. There are no variables for this procedure.

AddNewRequest Procedure
This procedure accepts data to be saved into the Open Requests file, and is one of the most commonly used procedures in the system. All data is entered into the system manually, with the exception of a unique Request ID which is generated by the system using data gained from the LoadIDs procedure. The following variables are used:

reqID	The request's unique ID
reqDateDay	The date on which the request was first reported. The date is separated into three parts, as it is automatically gathered from the system clock.
reqDateMonth	
reqDateYear	
reqCurrentStatus	The current status of the request, in numerical form.
reqUserForename	The full name of the user who originally reported the request.
reqUserSurname	
reqUserDept	The department of the user who originally reported the request.
reqSystemLoc	The location of the system which the request refers to.
reqSystemName	The name of the system which the request refers to.
reqErrorCat	The category of error which the system is reported to have.
reqErrorDesc	A more detailed description of the error.
reqNotes	Miscellaneous notes about this request.
reqAssignedTech	The initials of a technician assigned to a request

UpdateRequestStatus Procedure

This procedure is used by technicians to update the current status of a procedure once actions have been completed upon it. Additionally, technicians can leave notes with the status update. The following variables are used:

EnteredRequestId	The RequestID entered by the user when the procedure is first called.
RequestStatusUpdateOption	An integer to decipher whether or not a user wants to update the status of a request.
AddNotesOption	An integer to decipher whether or not a user wants to add notes to the request.

FindRequestById Procedure

This procedure is used by users to view the original details of a request by inputting the request's unique ID. The following variables are used:

userId	The Request ID entered by the user when the procedure is first called.
searchPosition	The position at which the search is at before a matching

	record is found.
foundRecord	A boolean variable to declare whether a record matching the criteria has been found.
SubMenuOption	A sub-menu option integer.
X	Declares the horizontal position of text.

FindRequestBySurname Procedure

This procedure is used by users to view the original details of a request by inputting the request user's surname. The following variables are used:

userSurname	The Request ID entered by the user when the procedure is first called.
searchPosition	The position at which the search is at before a matching record is found.
foundRecord	A boolean variable to declare whether a record matching the criteria has been found.
SubMenuOption	A sub-menu option integer.
X	Declares the horizontal position of text.

FindRequest Procedure

This procedure is a menu, with two options to choose between. Users can search for a request either by surname or by ID. One variable is used:

menuOption	Loads FindRequestBySurname or FindRequestById based upon the user's menu choice.

FindEmployeeById Procedure

This procedure is used by users to view details of an employee by inputting the employee's unique ID.

userId	The Request ID entered by the user when the procedure is first called.
searchPosition	The position at which the search is at before a matching record is found.
foundRecord	A boolean variable to declare whether a record matching the criteria has been found.
SubMenuOption	A sub-menu option integer.
X	Declares the horizontal position of text.

FindEmployeeBySurname Procedure

This procedure is used by users to view the original details of an employee by inputting the request user's surname. The following variables are used:

userSurname	The Request ID entered by the user when the procedure is first called.

searchPosition	The position at which the search is at before a matching record is found.
foundRecord	A boolean variable to declare whether a record matching the criteria has been found.
SubMenuOption	A sub-menu option integer.
X	Declares the horizontal position of text.

FindRequest Procedure

This procedure is a menu, with two options to choose between. Users can search for a request either by surname or by ID. One variable is used:

menuOption	Loads FindRequestBySurname or FindRequestById based upon the user's menu choice.

FindAssetById Procedure

This procedure is used to find an asset's details, when a user enters the asset's unique ID number. It is called from within the FindAsset procedure, and uses the following variables:

userId	The Asset ID entered by the user when the procedure is first called.
searchPosition	The position at which the search is at before a matching record is found.
foundRecord	A boolean variable to declare whether a record matching the criteria has been found.
SubMenuOption	A sub-menu option integer.
X	Declares the horizontal position of text.

MakeFiles Procedure

The MakeFiles procedure is used to create the Requests, Assets, IDData and Employees files when the program starts, if they do not already exist. It does not use any variables.

AddNewEmployee Procedure

This procedure accepts data to be saved into the Employees file, and is one of the most commonly used procedures in the system. All data is entered into the system manually, with the exception of a unique Employee ID which is generated by the system using data gained from the LoadIDs procedure. The following variables are used:

empId	A unique Employee ID, generated automatically after the employee's details have been entered for the first time through the AddNewEmployee procedure.
empForename	The employee's full name, stored in its forename and surname component parts to allow the system to use certain sections of the name at certain times – ie, to be more personal.
empSurname	
empDept	The department which the employee belongs to.

empPhone	A phone number for this employee.

AddNewAsset Procedure

This procedure is used to add a new asset to the system. It uses the following variables:

astID	A unique ID for this asset
astType	Type of product this asset is – eg workstation, server etc.
astMfgr	Asset's manufacturer
astModel	Asset's model number/name
astLoc	Location of asset within school
astPurchDate	Purchase date of asset (if known)
astPurchRetl	Purchase retailer of asset (again, if known)
astWarrExpir	Date on which warranty expires

DeleteRequest Procedure

This procedure is used to delete a request from the system, and uses the following variable:

RequestID	The ID of the request to be deleted, entered by the user

writeRequestsMenu Procedure

This procedure is used to write up the menu options contained within the "requests" submenu, and uses the following variable:

menuOption	The option entered by the user to navigate to various procedures from within the submenu.

writeAssetsMenu Procedure

This procedure is used to write up the menu options contained within the "assets" submenu, and uses the following variable:

menuOption	The option entered by the user to navigate to various procedures from within the submenu.

writeEmployeesMenu Procedure

This procedure is used to write up the menu options contained within the "employees" submenu, and uses the following variable:

menuOption	The option entered by the user to navigate to various procedures from within the submenu.

setPassword Procedure

This procedure is used to set a password for various procedures which hold sensitive information, such as the Statistics submenu. It uses the following variables:

userOldPassword	The existing password, as entered by the user. The user must enter this password in order to be able to change the password to something more memorable.
oldPassword	The existing password, as known by the system. If the value of this variable matches that of userOldPassword, then the user is allowed to change the password.
newPassword	The new password entered by the user. The user can only enter a new password if they successfully entered the existing password correctly.
newPasswordVerify	A quick way to test for any typographical discrepancies in the new password: ask the user to enter the password again. If newPassword and newPasswordVerify match in terms of their values, the password is stored in oldPassword and the user is presented with a success message. If they don't match, the user is taken back to entering the newPassword again until newPassword and newPasswordVerify match in terms of their values.

areYouSureQuit Procedure

The areYouSureQuit procedure confirms with users that they would like to quit before the system shuts down. This is useful as it may be fairly easy for a user to press the quit command accidentally. The variables used are:

quitAnswer	When asked whether the system should quit, this variable holds the answer that the user gives in boolean form.
quit	This value is updated to reflect the value of quitAnswer, and is used to control the outcome of the value stored in quitAnswer.

GlobalSettings Procedure

The GlobalSettings procedure allows the user to set how the program operates on a global basis. Generally, this procedure contains functions which fundamentally alter how the system processes data, and which sections of the system are open to all users. It uses two variables:

menuOption	Used to process the menu option that the user enters
AreYouSure	This variable is used to confirm that the user wishes to delete all of the data saved in the system-generated files.

viewTracking Procedure

The viewTracking procedure allows the user to quickly track the current status of any request stored in the system, as well as view the notes associated with a request. It uses the following variables:

TrackingRef	The tracking reference, as entered by the user.
searchPosition	The position at which the search is at before a matching record is found.
foundRecord	A boolean value to declare whether a matching record has been found.
SubMenuOption	The submenu option that the user enters. In this case, it is used to choose between viewing the original request details or the current status of a request.

Agenda procedure

The agenda procedure allows a technician to see all of the jobs that they need to complete in a tabular form. It uses the following variables:

enteredInitials	Initials of technician as entered by the user
X	Used as part of gotoxy to place text in certain positions on the screen in a neat form.
Y	

writeMainMenu procedure

This procedure writes the main menu, from which all other submenus are accessible. The following variables are used:

menuOption	The submenu option that the user enters. In this case, it is used to choose between the various submenus on offer.
quitOpt	A boolean value to gather whether to allow the system to quit

Good last plan covering all aspects of system. Most test data clear. Only one example of extreme test data.

TESTING

TEST PLAN

Test No	Screen	Purpose	Test Data	Expected Result	Type	Result
1	Main Menu	Confirm that it is possible to access each of the sub-menus by typing the corresponding number in to the system.	Entering numbers in to the main menu when prompted to access submenus	The sub-menus will load without problem, and more options related to each sub-menu will then be shown to the user.	Typical	✓
2	Main Menu	Check that it is possible to quit the program entirely, ending up within Turbo Pascal, by using the Quit menu option.	Attempting to quit the program entirely using the quit procedure.	The program should prompt the user to confirm that they wish to quit. Upon confirming the user wishes to quit, Turbo Pascal should be shown.	Typical	✓
3	Requests Menu	Check that it is possible to return to the main menu using the corresponding option.	Navigating to a submenu then navigating to the main menu using only the options supplied within the program.	The program should return the user to the main menu when the 'Return to main menu' option is selected.	Typical	✓
4	Main Menu	Check that pressing a number which does not correspond to a menu option does not result in an error, nor does it launch a procedure	Press a number which does not correspond to a submenu option	The program should accept and ignore the entered number, and await entry of a number which does	Erroneous	✓

		or function.		correspond to a menu option		
5	Add Request	Check that the Forename field accepts a maximum of twenty characters.	Use a forename which is 21 characters long	The program should allow a forename of twenty characters to be entered, however this is on the acceptable boundary.	Extreme	✓
6	Add Request	Check that the date which is automatically entered in the "Today's Date" field is correct.	Check that the date recognised by the system is the correct date	The date entered should be the same as the date in the system clock.	Typical	✓
7	Add Request	After entering information relating to the job request, the system should return a summary screen of all information accepted about a request.	Enter a new job request	The summary screen should be shown with the correct information displayed for the request in question.	Typical	✓
8	Add Request	The reference number provided for the request should not already exist.	Use a false reference number	The system should use the LoadIDs procedure to load the last-used ID into the system, and then create the reference number for the new request sequentially.	Typical	✓
9	Add Request	The job reference for the last request added should be saved to the iddata.dat file when the program is quit.	Open up iddata.dat to check the data is saved there.	The system should save the details of the last reference number to the iddata.dat	Typical	✓

				using the SaveIDs procedure when the program quits.		
10	View Request by ID	Entering a request ID which does not exist.	Use a false request ID	The system should accept the request ID, but should not show any information relating to a request.	Erroneous	✓
11	View Request by ID	Allowing the system to show multiple search results when a Request ID is entered.	Create a new request with the same request ID by entering its details into openrequests.dat directly.	Showing multiple results when searching for request details by Request ID.	Erroneous	✓
12	Update Request	Prompting the user to provide details about the technician currently dealing with a particular request when the Update Request procedure is run for the first time for a particular request.	Provide the technician's initials – eg ABC	The system should draw in specific details about the request in question, and use them in the way it approaches the user to ask for information about the technician. Additionally, it should also ask the user to enter the initials of the technician currently dealing with the request.	Typical	✓

13	Update Request	Prompting the user to provide details about the technician currently dealing with a particular request when the Update Request procedure has already been run for a particular request.	Provide the technician's initials	The system should realise that the request has already been assigned to a technician and not allow the user to enter a technician's initials.	Erroneous	✓
14	Update Request	Allowing the user to update the current status of a request that already exists in the system.	Providing a way to update a request	The system should allow the user to update the current status of any request in the system at any time.	Typical	✓
15	Update Request	The system should allow the user to choose between seven options to choose between to update the current status of a request: 1. Open 2. Under investigation 3. Awaiting delivery of parts 4. Being resolved 5. Requires attention 6. Ready for collection 7. Other	Asking the system what status their job has got to	The system should show the options noted to allow the user to update the request. Once an option has been submitted, this data should be saved to the relevant point in the Requests file.	Typical	✓
16	Update Request	The user should be asked whether they wish to input notes into their update. Notes can constitute any information not covered by the	Prompting user to input notes – with Yes/No choice.	The user should be prompted to add notes, and given two choices: yes and no. If no is entered, the	Typical	✓

		seven rather broad topics above.		system should return the user to the main menu. If yes is entered, the user should be able to enter a short set of notes about the update, which are then stored in the Requests file at the relevant point.		
17	Update Request	A confirmation screen should appear.	Updating a request, then checking to see if the summary screen appears	The summary screen should report to the user any changes which have been made to a request, and have – ultimately – been saved to the Requests file.	Typical	✓
18	View Request by Surname	Entering a surname for which no requests exist under.	Enter a surname which does not relate to a request	The system should accept the details, and return the user to the main menu.	Typical	✓
19	View Request by Surname	Entering a surname for which many requests exist under.	Enter a surname which does not relate to a request	The system should print out each request, one at a time, in full.	Typical	✓
20	Delete Request	The system asks for the system password, followed by the Request ID of the request to be deleted.	Enter a request ID when asked.	The system should delete the request in question, and confirm this to the user.	Typical	✓
21	Delete Request	The system asks for the Request ID of the request to be deleted, but the	Enter a request ID that does not	The system should inform the user that deletion of	Typical	✓

		Request ID cannot be found.	exist	that particular request cannot be completed because it does not exist.		
22	Add Asset	The system should ask the user for a type of asset, under six main categories: • Workstation • Server • Whiteboard • Projector • Peripheral • Other	Tested by adding a new asset to the system.	The system asks the user to categorise the asset into one of the six categories provided.	Typical	✓
23	Add Asset	The system should ask for the manufacturer of an asset.	Tested by adding a new asset to the system.	The system should accept input from the user for this particular field, and should save this to the Assets file.	Typical	✓
24	Add Asset	The system should ask for the model name of an asset.	Tested by adding a new asset to the system.	The system should accept input from the user, and save this to the Assets file.	Typical	✓
25	Add Asset	The system should ask the user to provide the location of this asset.	Tested by adding a new asset to the system.	The system should accept input from the user, and save this to the Assets file. Typically, data entered here will be in the form of a two-letter room name, however may also be longer if a location has no assigned two-letter name.	Typical	✓

26	Add Asset	The system should ask the user for the purchase date of this asset.	Tested by adding a new asset to the system.	The system should accept input from the user, but limit the allowed input to ten characters. Doing so limits the amount of storage space required to save an asset, whilst also allowing the user choice as to which format a date is entered in, such as dd/mm/yy or dd/mm/yyyy.	Typical	✓
27	Add Asset	The system accepts a date which is longer than ten characters.	Entering a date which is more than ten character in length – eg 01/02/2003456	The system should not allow this data to be entered, and should prompt the user to enter a date which is under ten characters in length.	Erroneous	✓
28	Add Asset	The system should prompt the user to input the purchase retailer of the asset in question.	Tested by adding a new asset to the system.	The system should accept this data, and process it in the usual way – by adding it to the relevant point in the Assets file.	Typical	✓
29	Add Asset	The system should ask the user for the warranty expiration date of this asset.	Tested by adding a new asset to the system.	The system should accept input from the user, but limit the allowed input to ten	Typical	✓

				characters. Doing so limits the amount of storage space required to save an asset, whilst also allowing the user choice as to which format a date is entered in, such as dd/mm/yy or dd/mm/yyyy.		
30	Add Asset	The system accepts a date which is longer than ten characters.	Tested by adding a new asset to the system.	The system should not allow this data to be entered, and should prompt the user to enter a date which is under ten characters in length.	Erroneous	✓
31	Find Asset by ID	The system should prompt the user for an asset's ID, and show details about that particular asset.	Tested by finding an asset in the system.	The system shows details about the asset corresponding to the asset ID inputted by the user.	Typical	✓
32	Find Asset by Type	The system should prompt the user to choose the type of asset, and should then provide information about all assets of a particular type.	Tested by finding an asset in the system.	The system shows details about the assets corresponding to the type submitted.	Typical	✓
33	Find Asset by Manufacturer	The system should prompt the user to enter the manufacturer of the asset, and should then provide information about all assets	Tested by finding an asset in the system.	The system shows details about the assets built by a particular manufacturer in tabular form.	Typical	✓

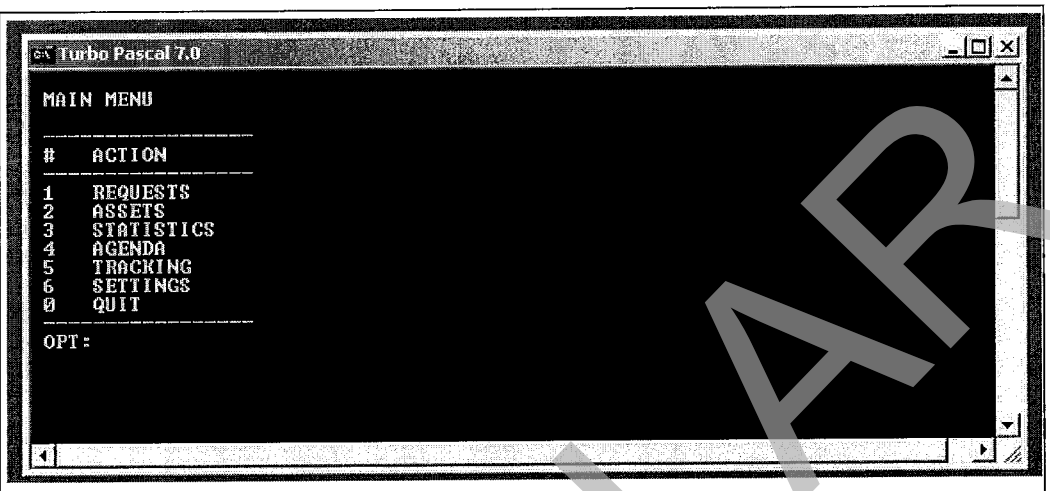
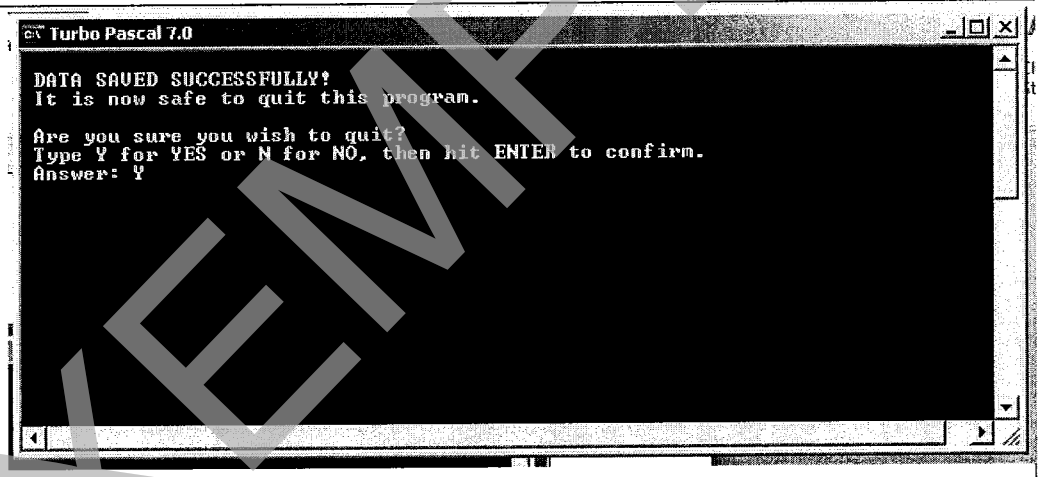
		manufactured by a particular manufacturer in tabular form.				
34	Find Asset by Location	The system should prompt the user to enter the location of the asset, and should then display information about all assets in a particular location, regardless of type and manufacturer.	Tested by finding an asset in the system.	The system shows details about the assets existing in a certain location in tabular form.	Typical	✓
35	Delete Asset	The system asks for the Asset ID of the asset to be deleted.	Tested by attemptin g to delete an asset in the system.	The system should accept the Asset ID entered, and return the user to the assets menu.	Typical	✓
36	Delete Asset	The system asks for the Asset ID of the asset to be deleted, but the Asset ID cannot be found.	Entered 57, an asset ID which does not exist.	The system should inform the user that deletion of that particular asset cannot be completed because it does not exist.	Typical	✓
37	Statistics	The system should generate live statistics about requests and assets "on the fly".	Tested by attemptin g to view statistics.	The system shows information in an easily accessible format about requests and assets.	Typical	✓
38	Agenda	The system should prompt the user to enter the initials of a technician.	Tested by attemptin g to view agenda.	The system should return the details about requests currently assigned to a particular technician.	Typical	✓

Could be clearer.

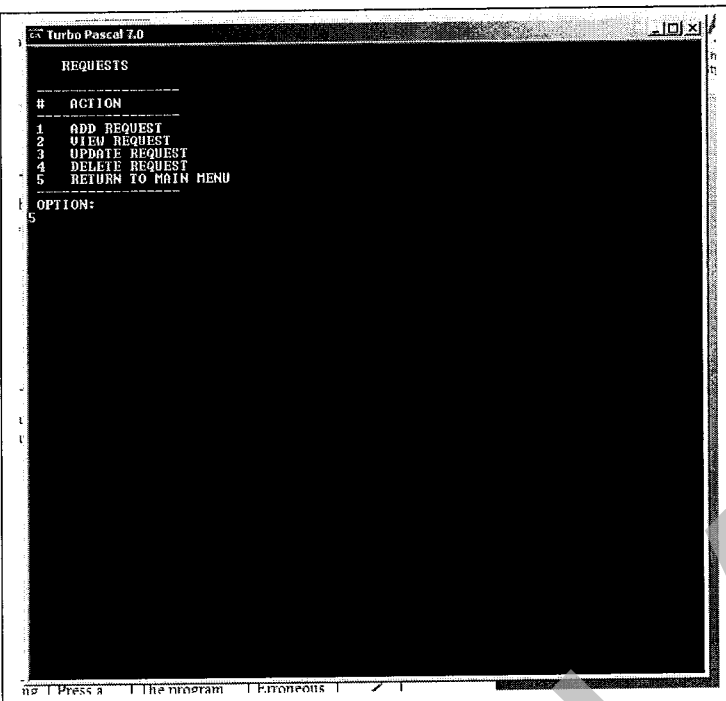
39	Agenda	The system prompts the user to enter the initials of a technician, but the technician has no request assigned to them.	Entering a valid technician's initials but who has had their request completed and deleted.	The system accepts the information, and returns the user to the main menu.	Typical	✓
40	Agenda	The system prompts the user to enter the initials of a technician, but the technician entered does not exist.	Entered fake technician's initials	The system doesn't display any details at all.	Erroneous	✓
41	Password	The system checks to see if a password already exists.	Defining a password before rerunning the Password procedure	If the system recognises that a password doesn't exist, it should prompt the user to create a password.	Typical	✓
42	Password	The system checks to see if a password already exists, and recognises that a password does exist.	Defining a password before rerunning the Password procedure	The system should prompt for the existing password, before allowing the user to change the password.	Typical	✓
43	Password	The system checks to see if a password exists, and should not allow access to a particular feature until the correct password is entered.	Defining a password, then attempting to access a feature restricted by a password	The system prompts the user for the password, and gives a little explanatory note explaining why the password is required.	Typical	✓

Clear record of testing, cross referenced to test plan. A couple of results require further explanation.

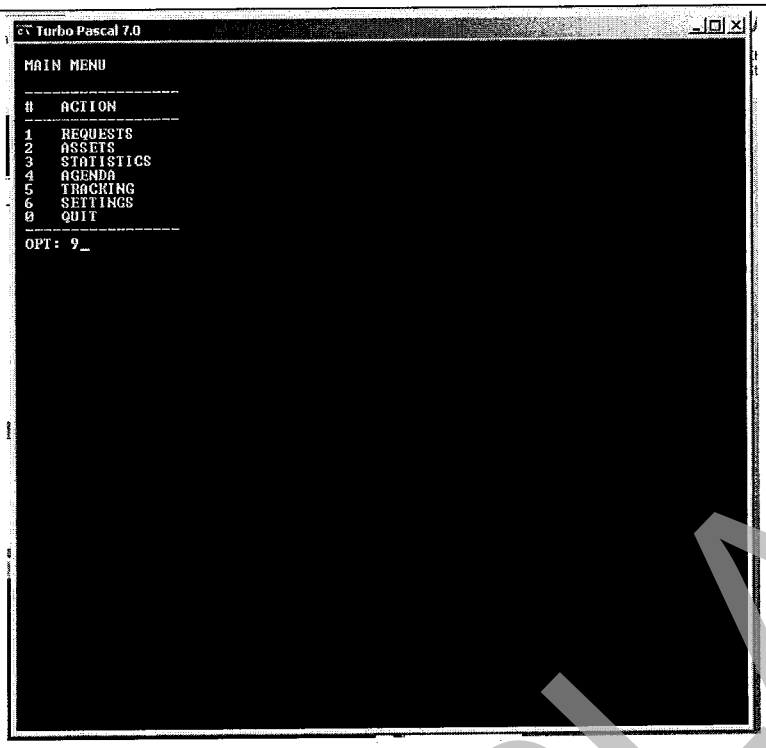
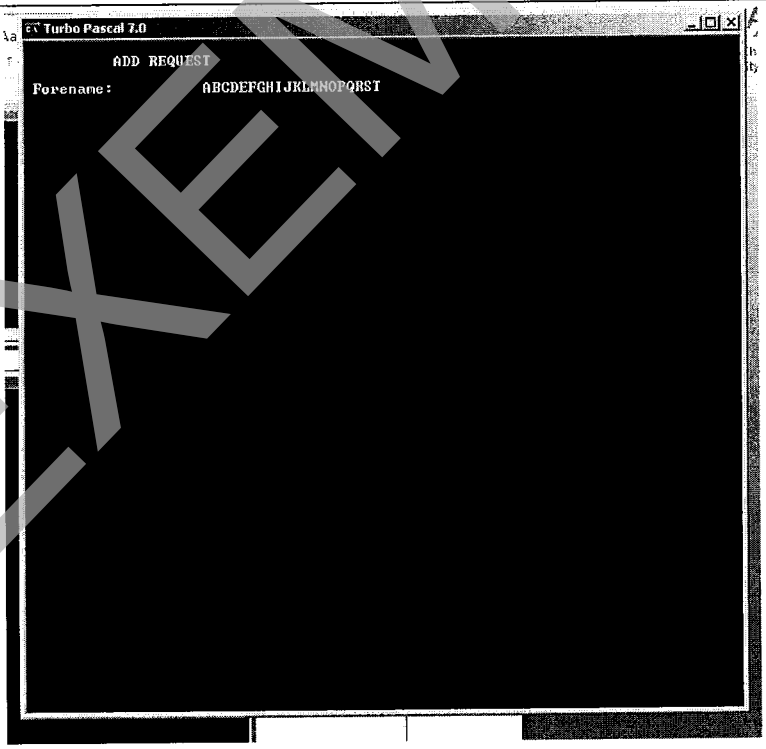
TESTING EVIDENCE

1	 In this test, I am confirming that it is possible to access each of the submenus from the main menu.
2	 In this test, I am checking that it is possible to quit the program entirely. This screenshot is of the areYouSureQuit procedure, which runs when the user asks the system to terminate itself from the main menu.

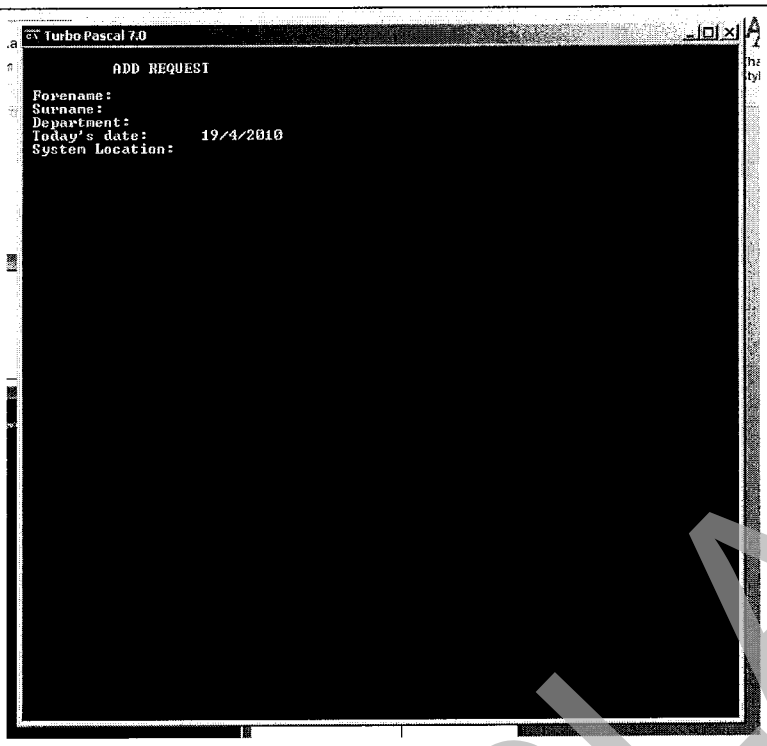
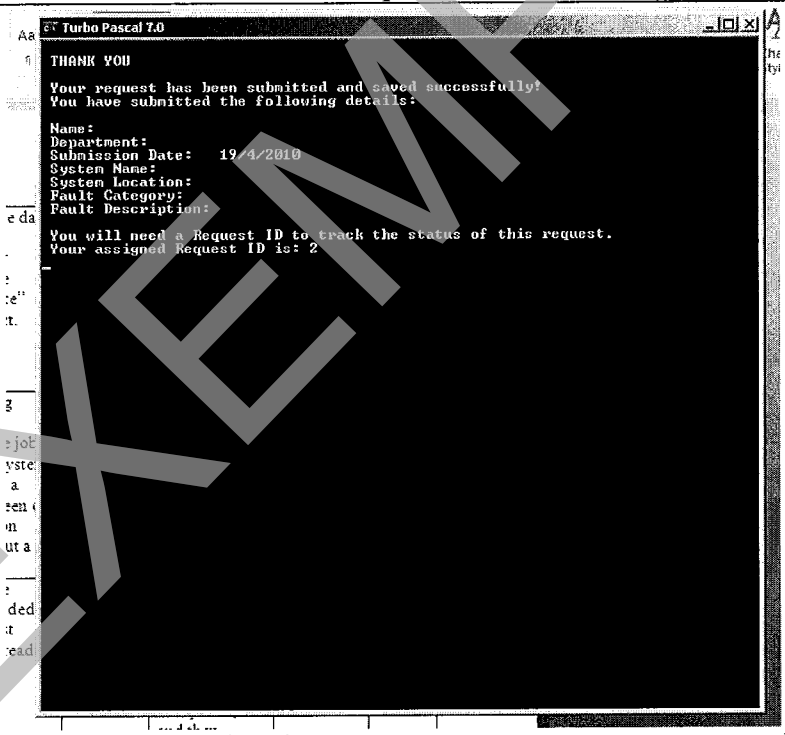
3

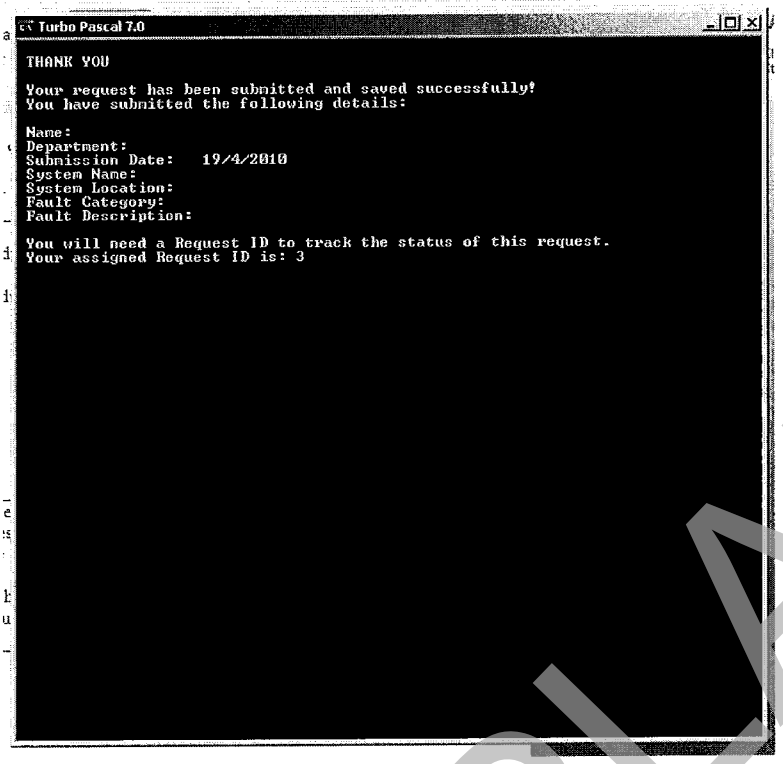
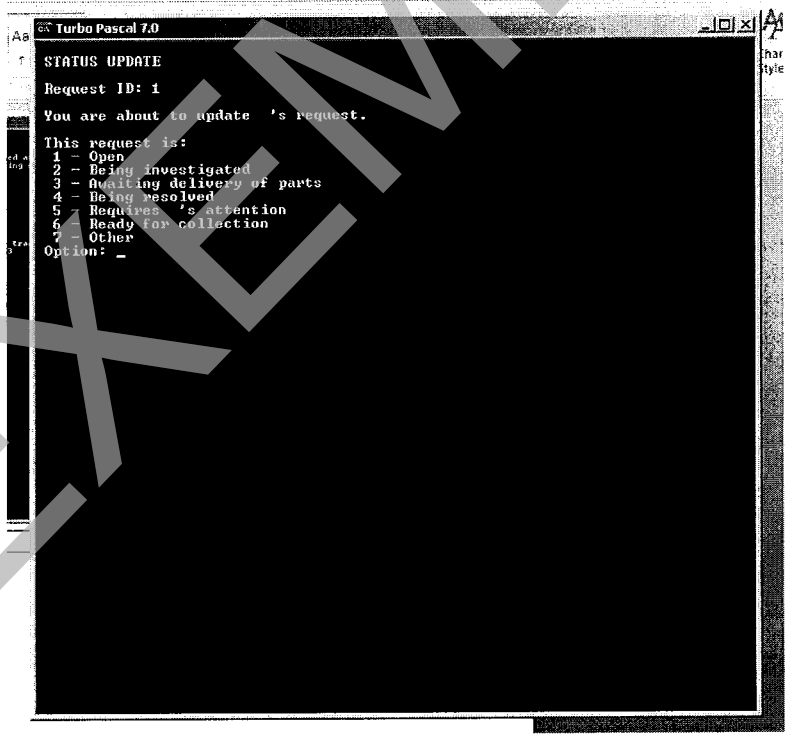


In this test, I am attempting to access the main menu from a submenu. It is clear that I can access the main menu from the Requests submenu by using option five in the Requests submenu.

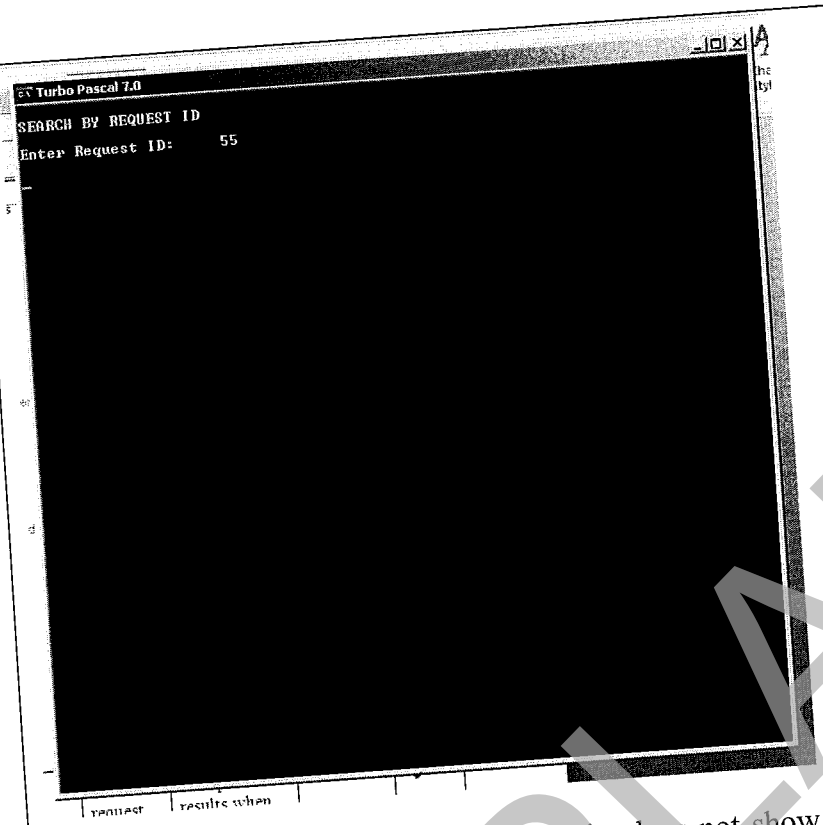
4	 In this test, I am attempting to check that pressing a number which does not correspond to a menu option does not result in an error, nor should it launch a procedure or function. By using option nine, as there is no corresponding option, the system should ignore the entered number and await entry of a number which does exist.
5	 This test checks that the forename field only accepts a maximum of twenty characters. Entering twenty-one characters should result in an error.

Not clear if passed.

6	 This test checks that the correct date has been gathered from the computer as the date in this section is computed automatically.
7	 This test checks that the system returns a summary screen when information is entered into the Add Request screen, and that the summary screen displays the correct information.

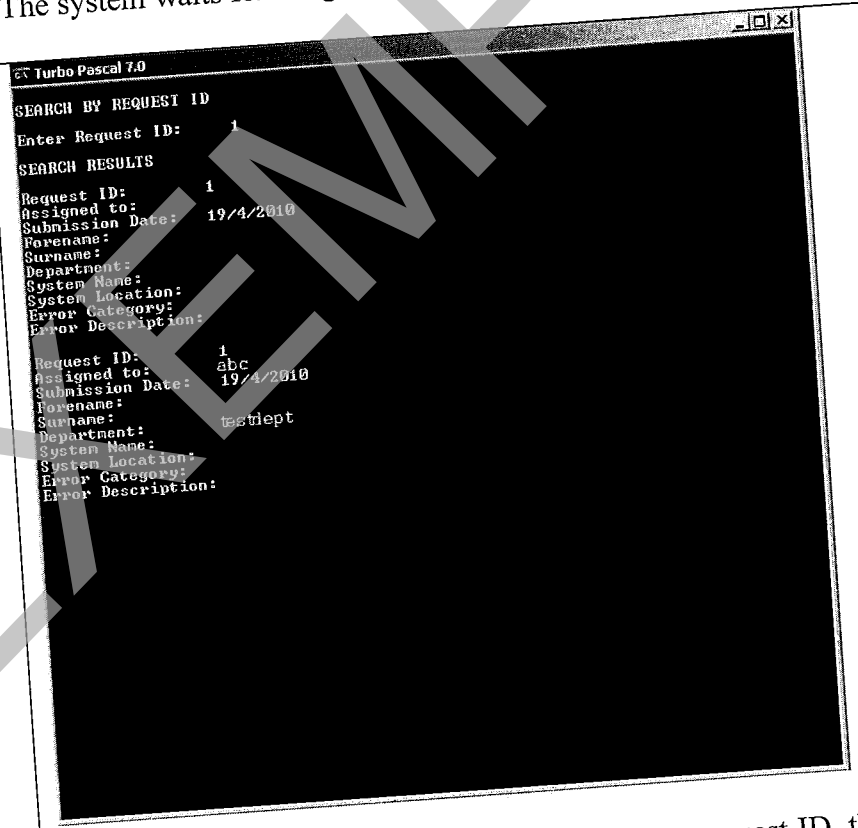
8	 This test is used to check that the request ID given does not already exist. If it did, the system would report an error and the user would then be told to enter their details again.
9	

10

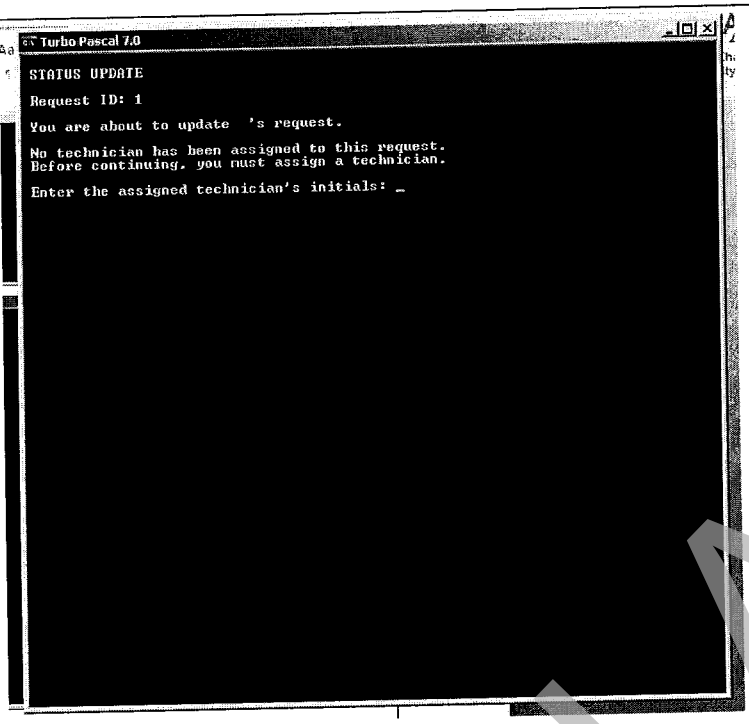
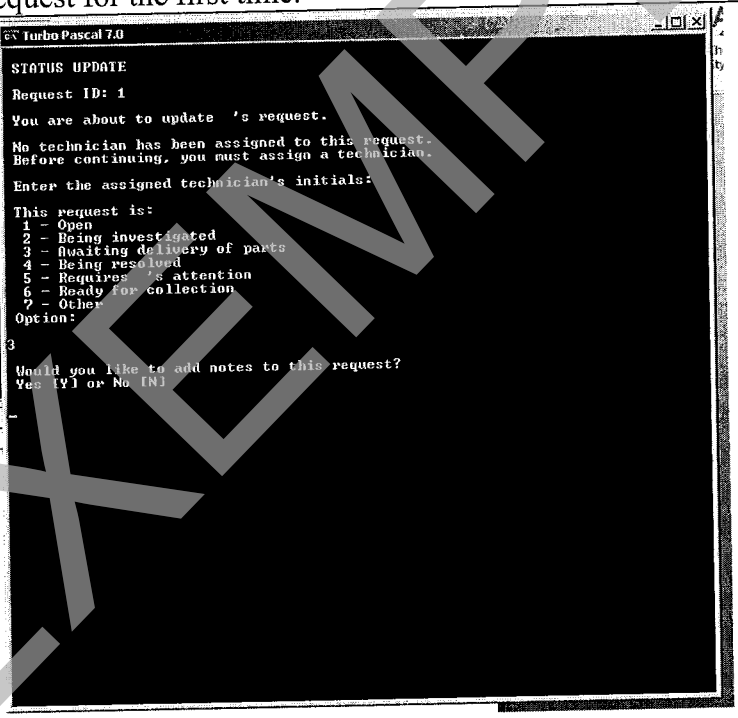


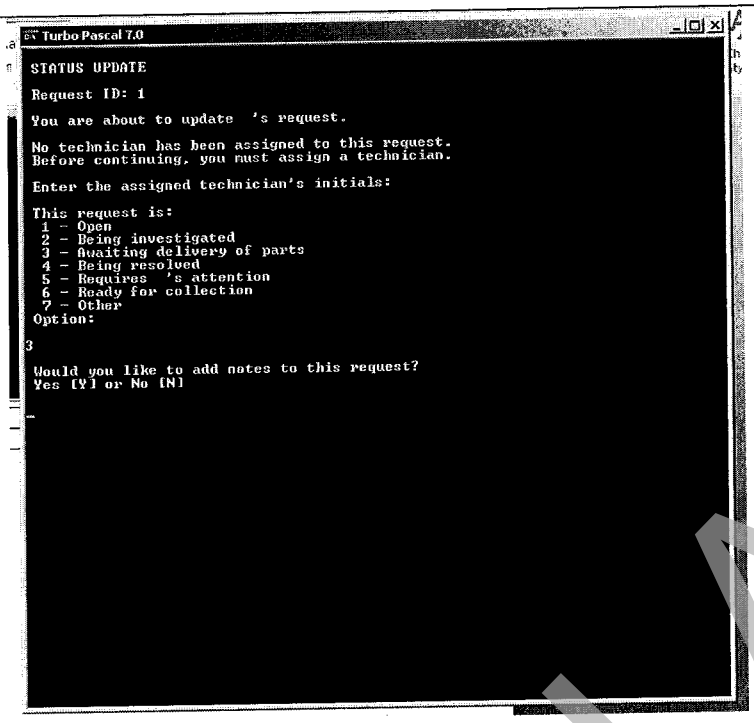
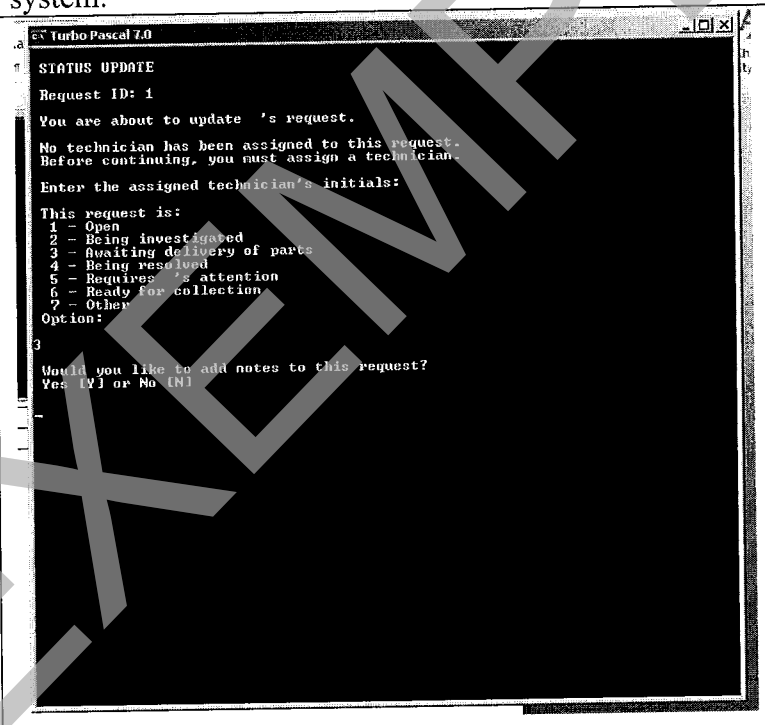
Entering a request ID which does not exist does not show any details at all. The system waits for a legitimate request ID to be entered.

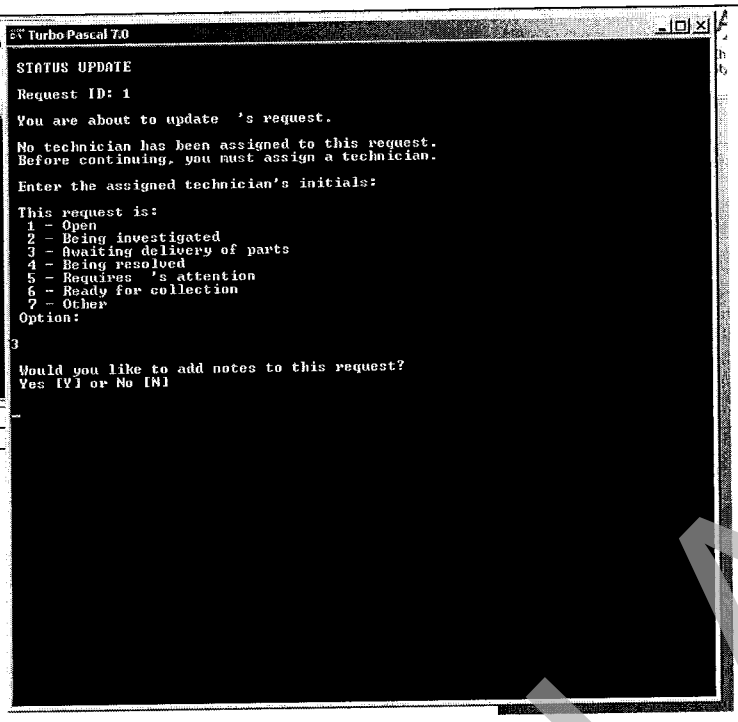
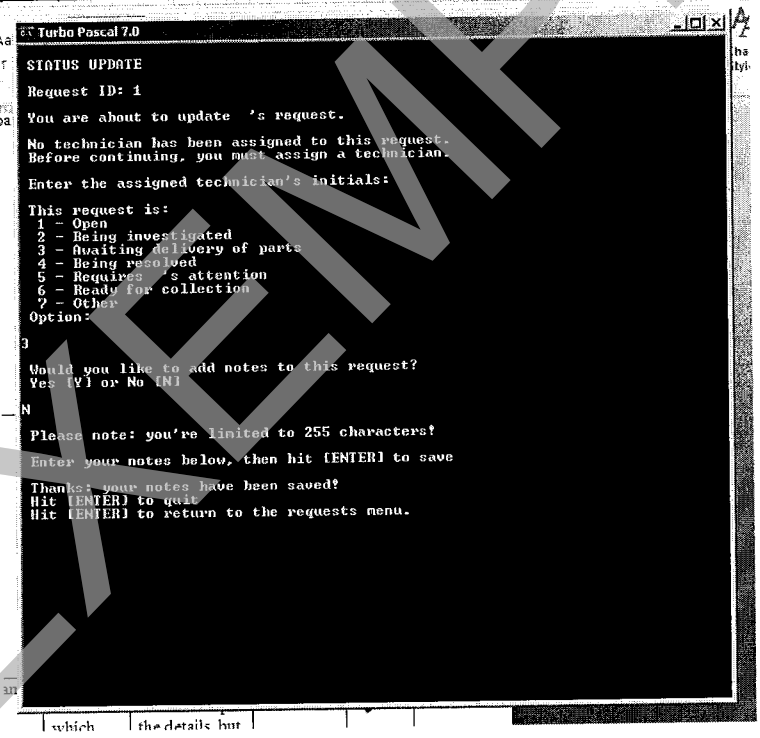
11



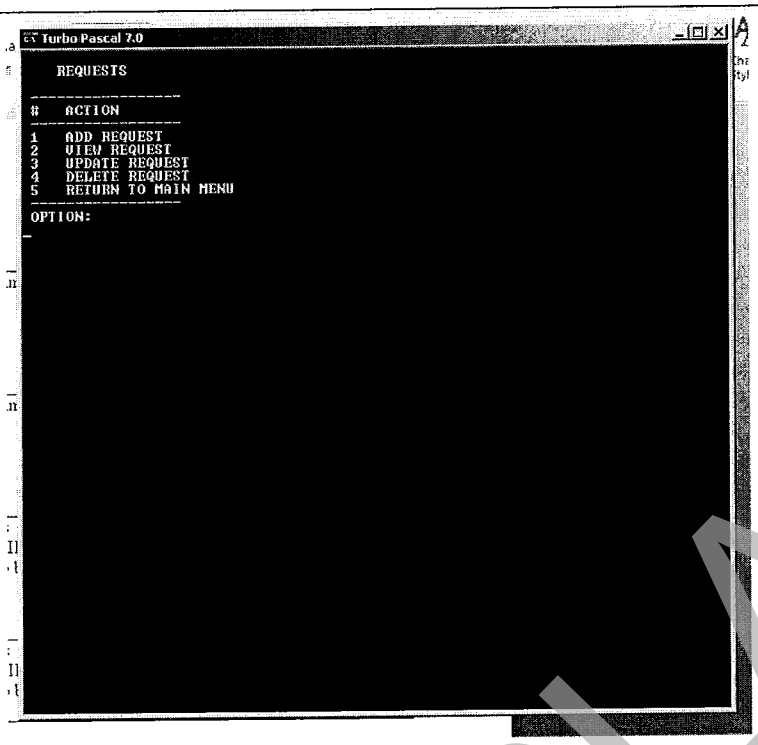
If multiple requests do happen to share the same request ID, then each request is shown independently of each other so that the user has clear and correct information available to them.

12	 The system prompts the user to enter a technician's initials when updating a request for the first time.
13	 The system realises that a technician has been assigned to a request and does not ask for a technician's details to be entered.

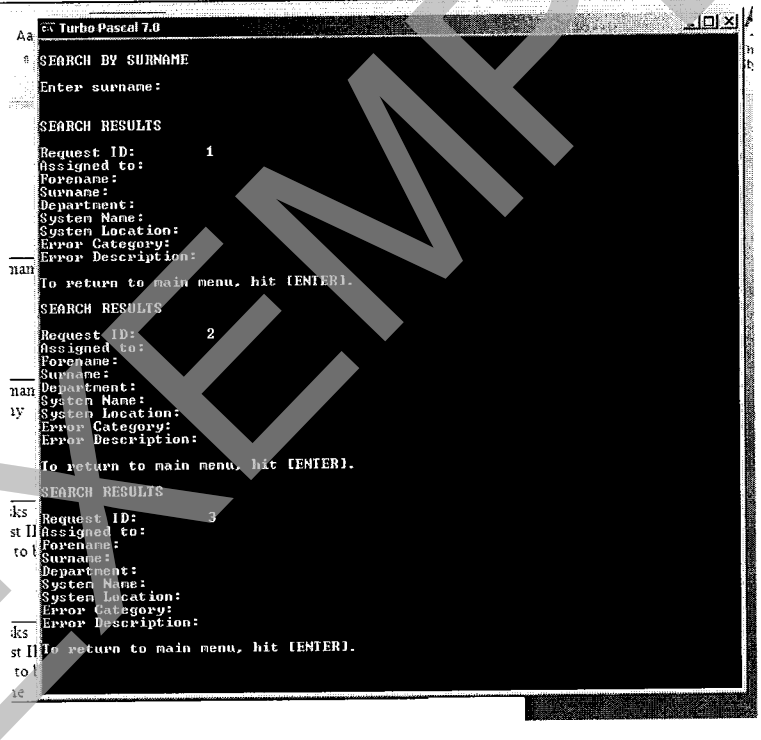
14	 The system allows the user to update the current status of any request in the system.
15	 The system should prompt the user to input where their job is up to.

16	 The user should be asked whether they want to enter any explanatory notes to this update.
17	 The system should allow the user to browse through all of the information entered before quitting.

18

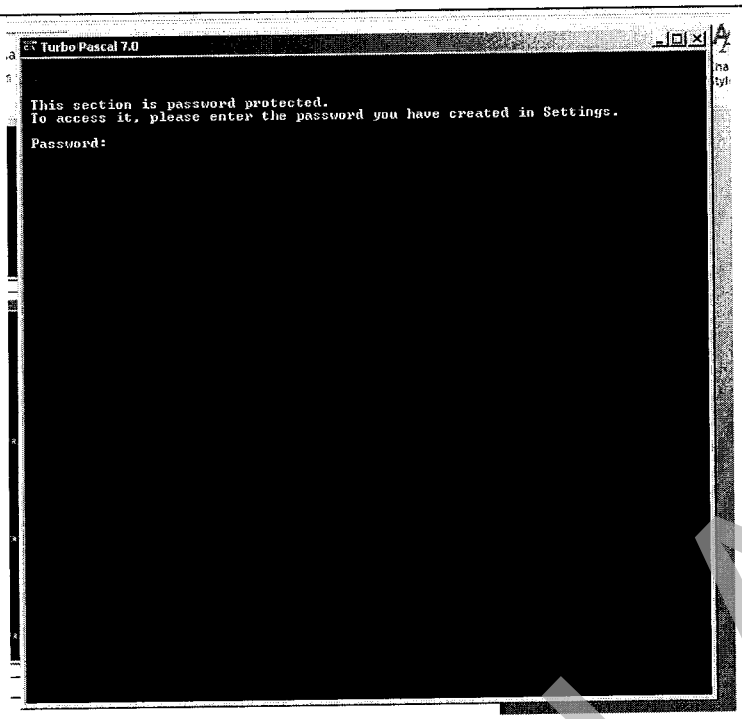


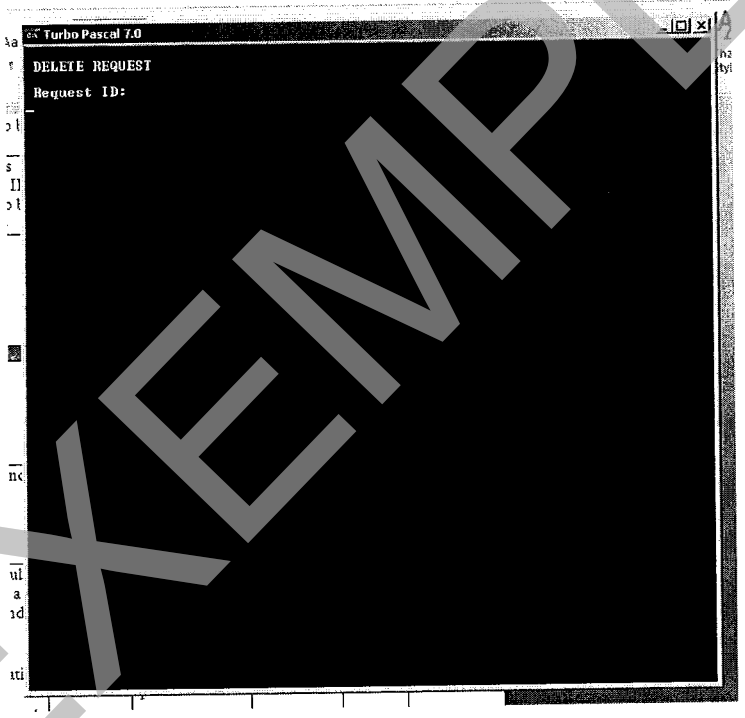
19



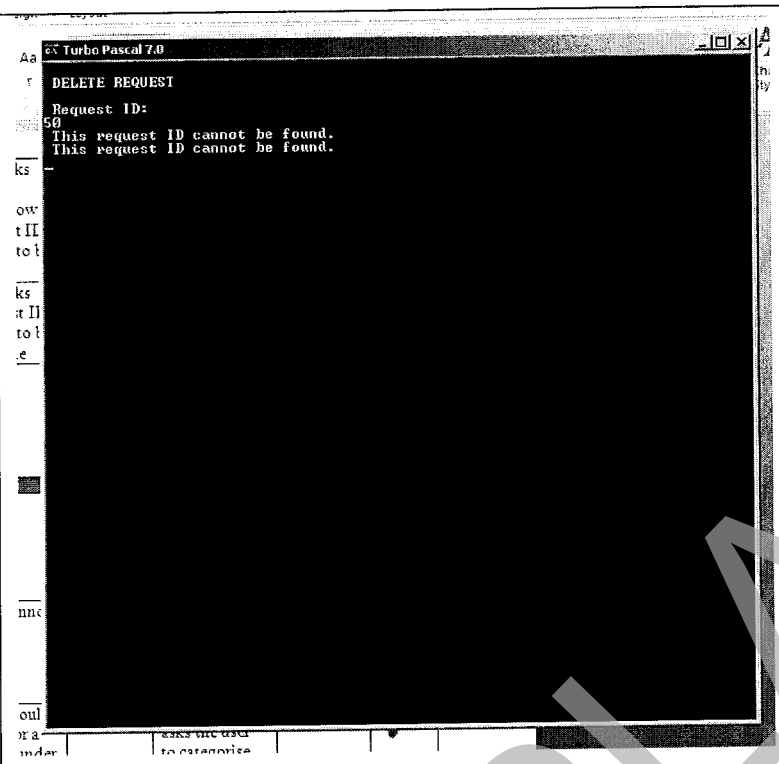
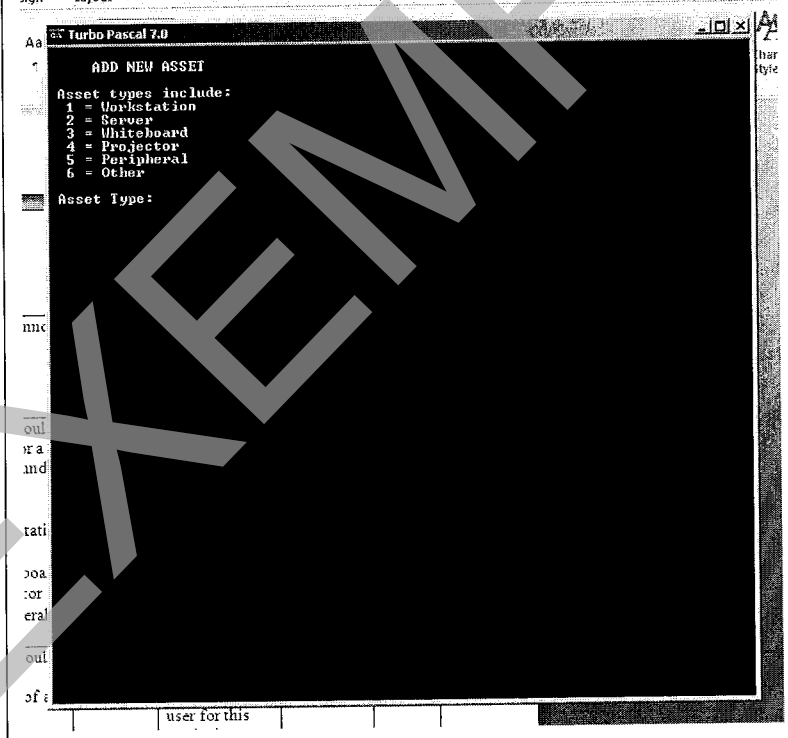
Entering a surname for which many requests appear under simply writes the details of each requesting matching the surname. This provides a clear, efficient way of displaying data to the user.

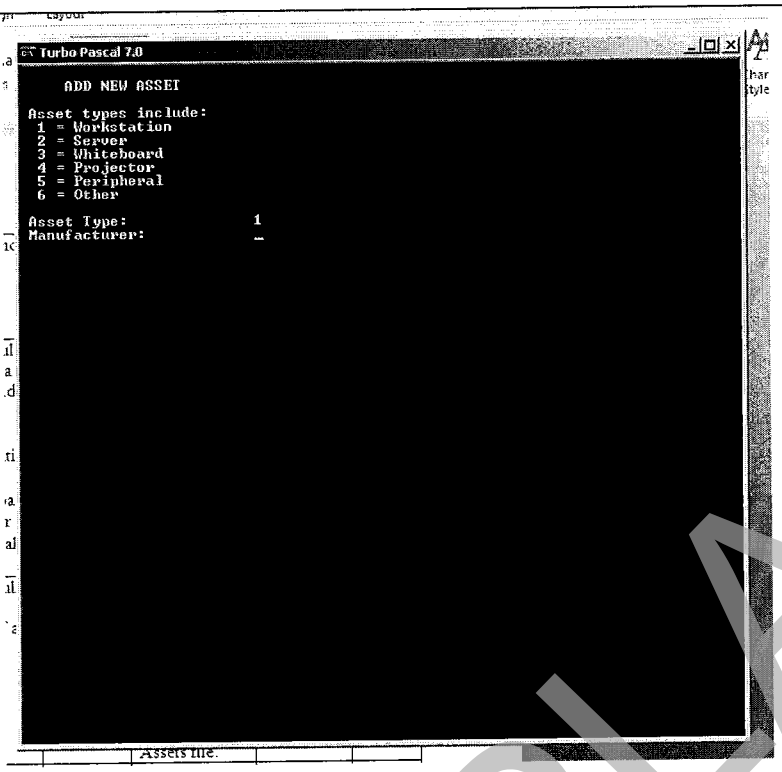
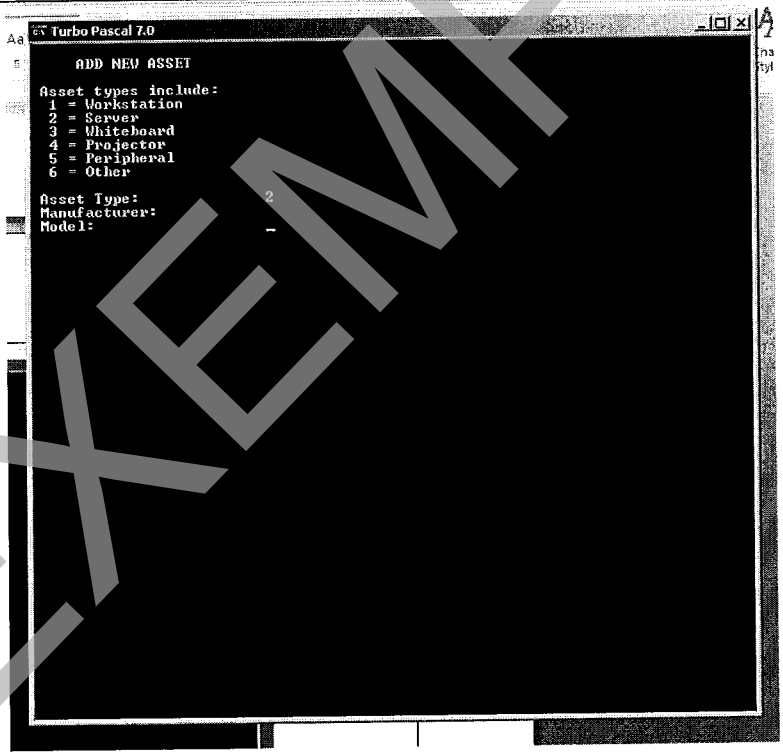
20

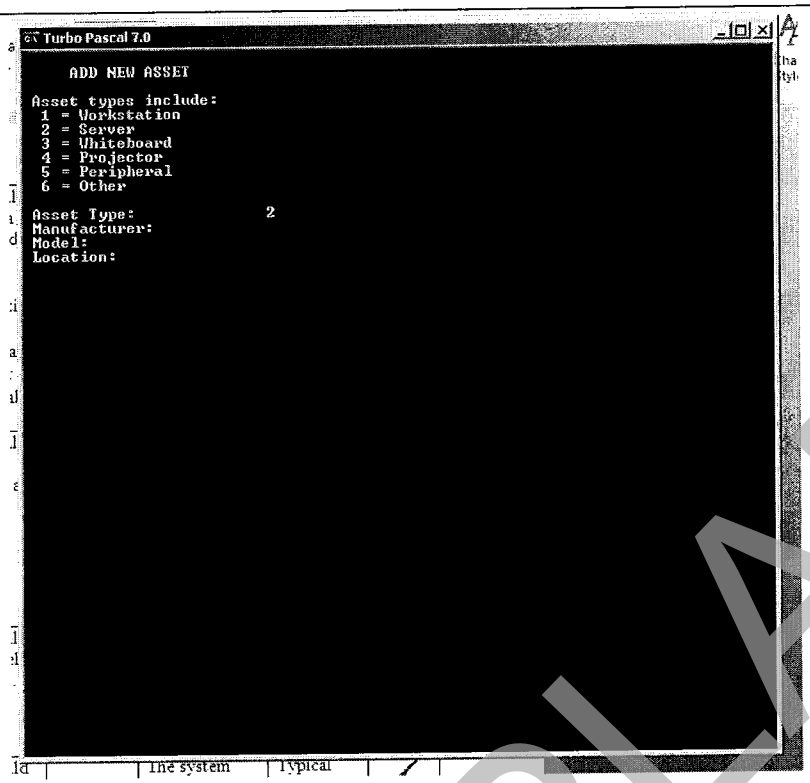


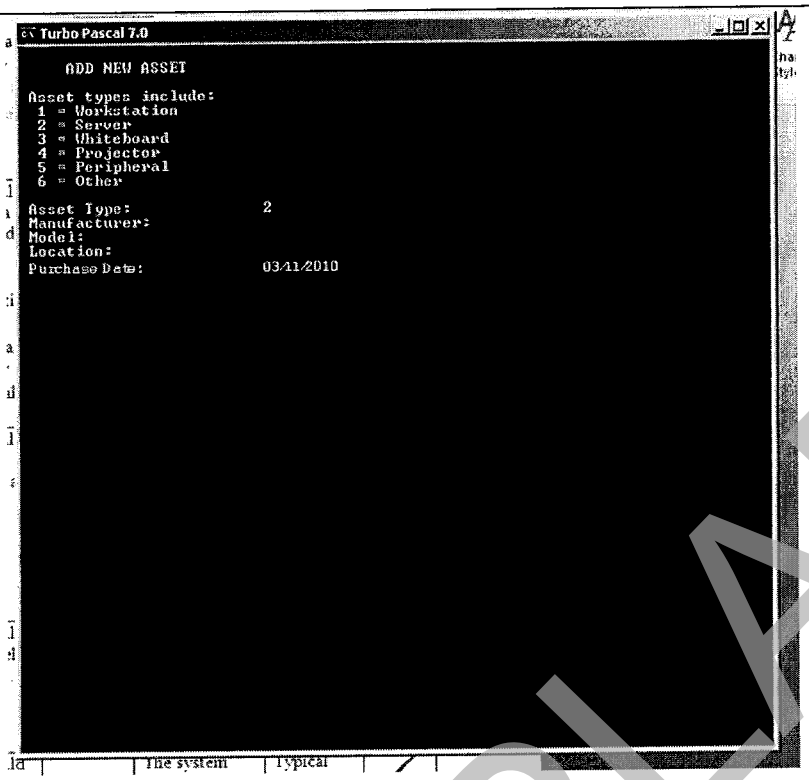


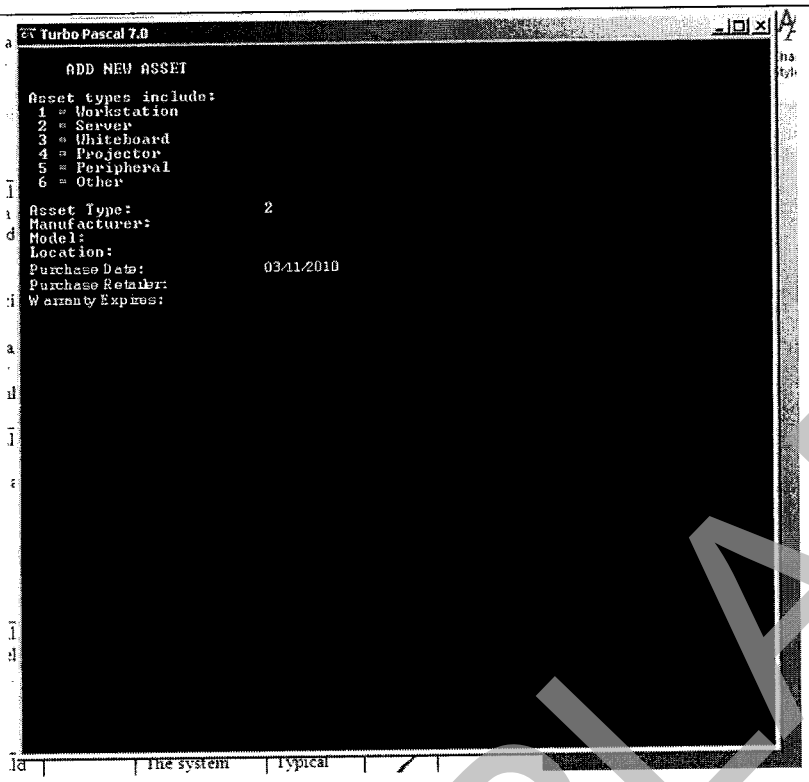
The system asks for a password before allowing the user access to the Delete Request procedure.

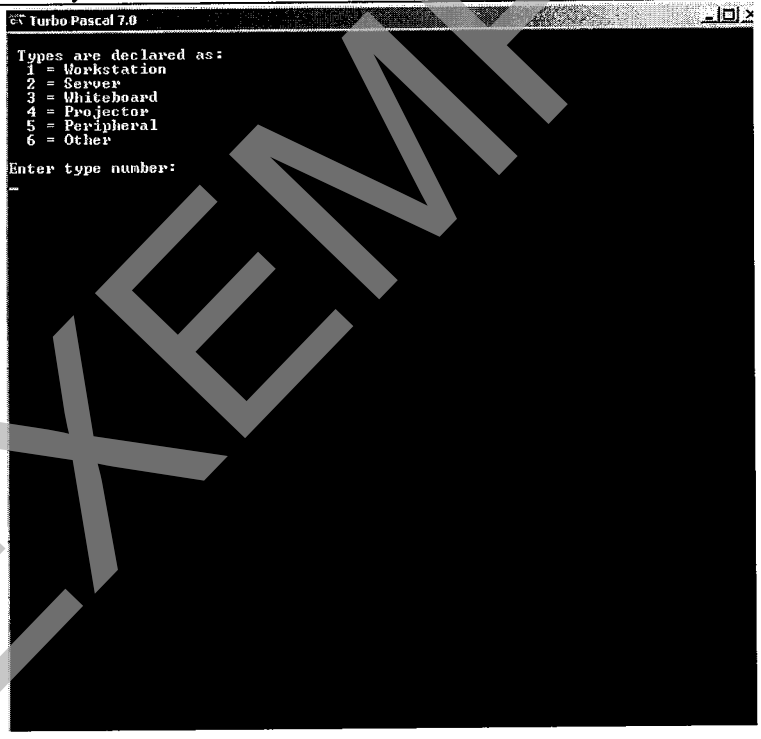
21	 The system asks the user for a request ID, but the system cannot find the entered request ID. The system therefore produces an error.
22	 The system asks what type of asset the user would like to add.

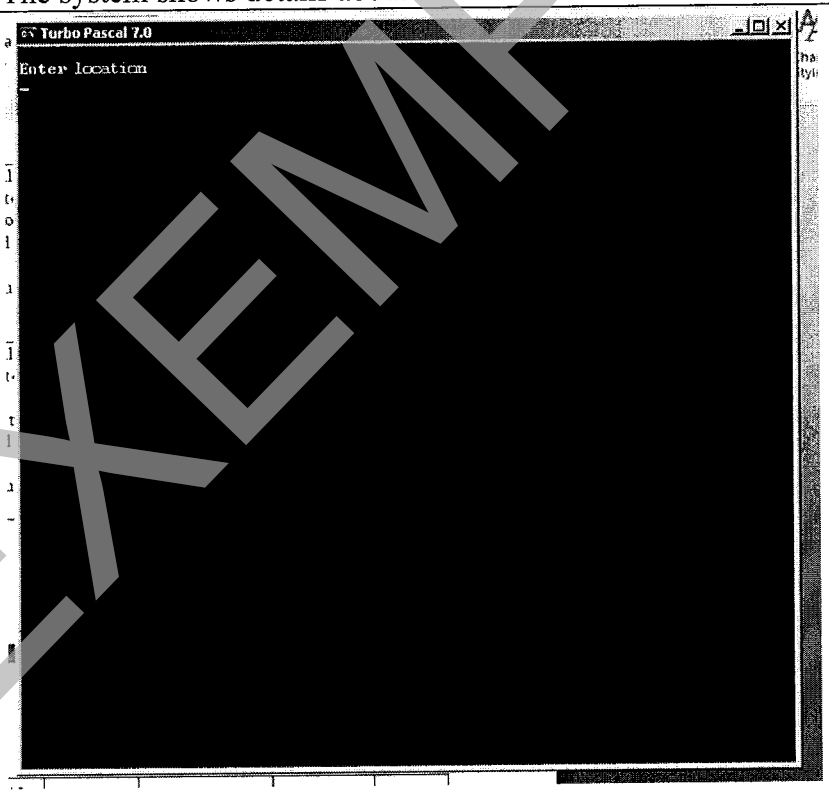
23	 The system asks for the asset manufacturer.
24	 The system asks for the asset model.

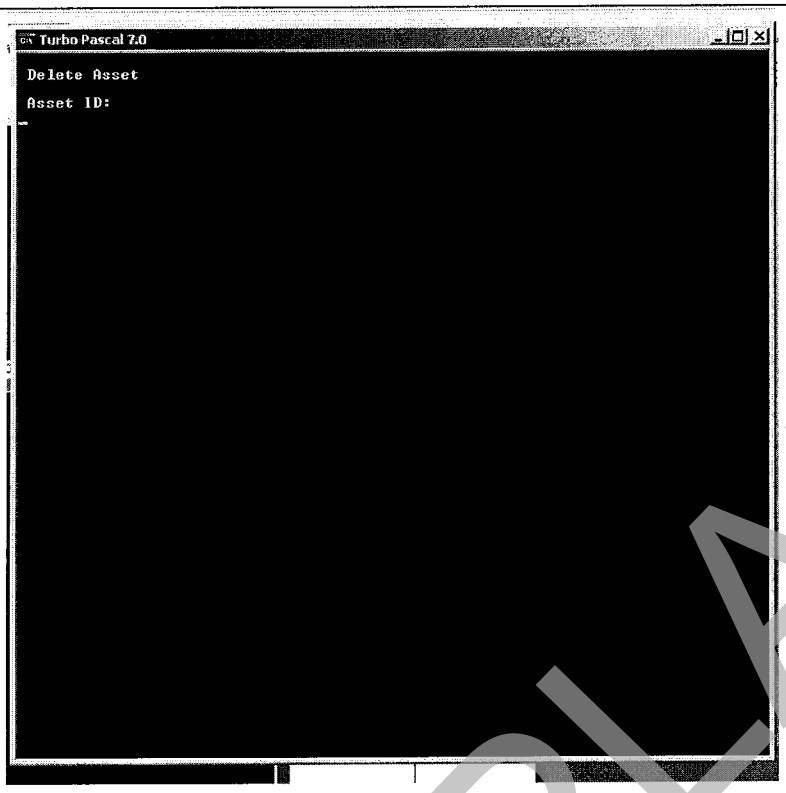
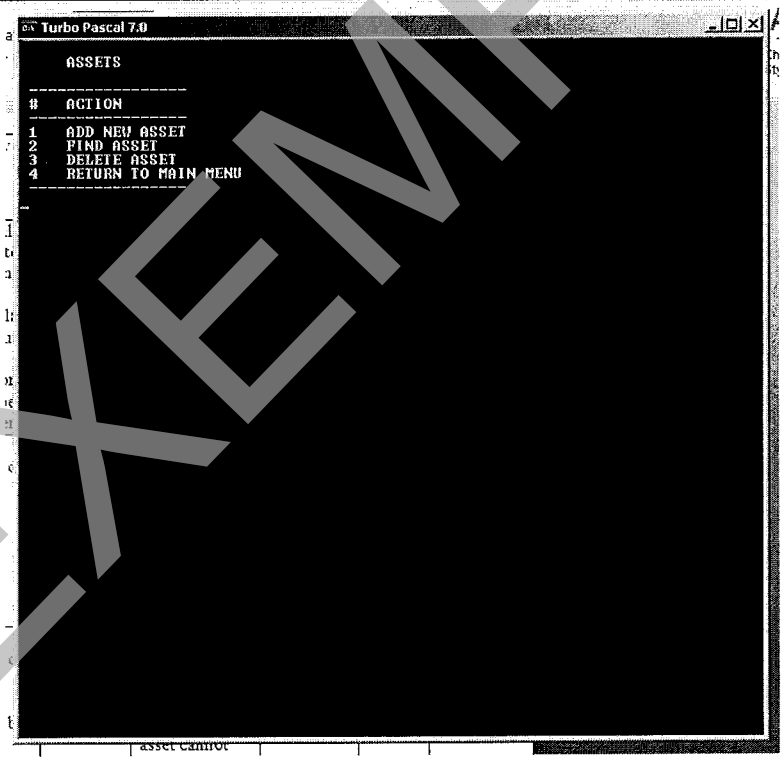
25	 The system asks for the asset's location.
26	 The system asks for the asset's purchase date.

27	 The system only accepts dates of less than ten characters.
28	 The system asks for the purchase retailer.

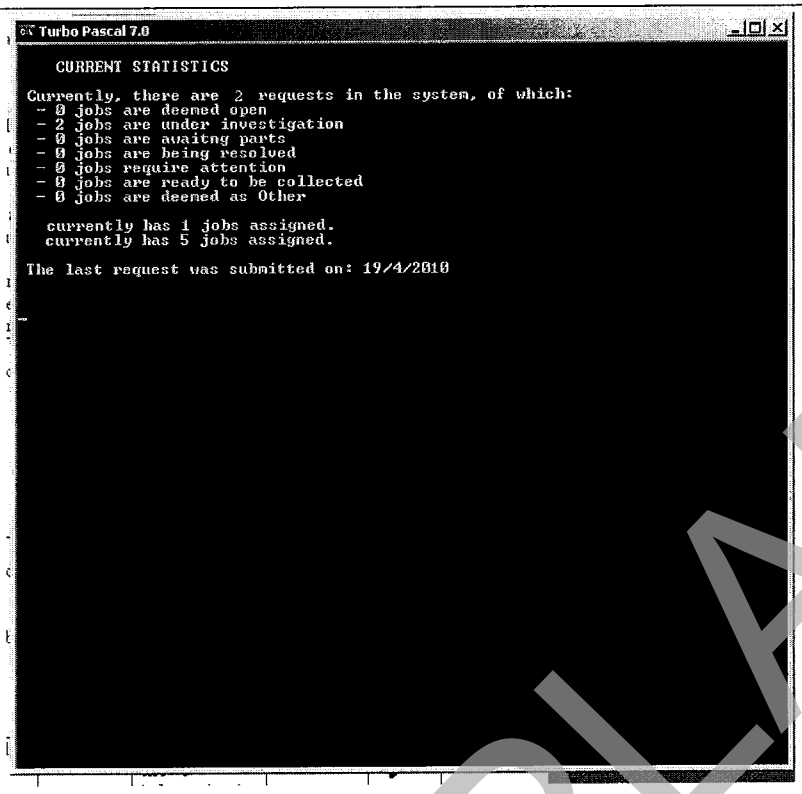
29	 The system asks for the date on which the warranty expires.
30	 The system accepts a date in the Warranty Expiration field which is more than ten characters.

31	 The system asks for the asset ID and then displays the details.
32	 The system shows details about an asset based on the type entered.

33	 The system shows details about an asset based on the manufacturer entered.
34	 The system asks for the asset's location.

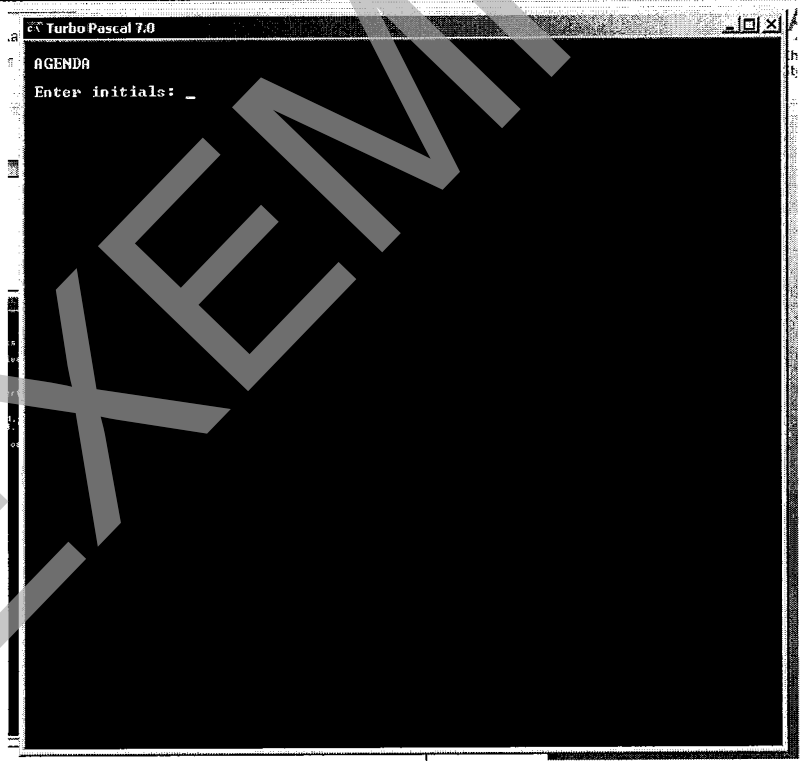
35	 The system asks for the asset's ID to be to deleted.
36	 The system asks foor the asset ID to be deleted, but the ID does not correspond to an asset. The user is returned to the Assets menu.

37

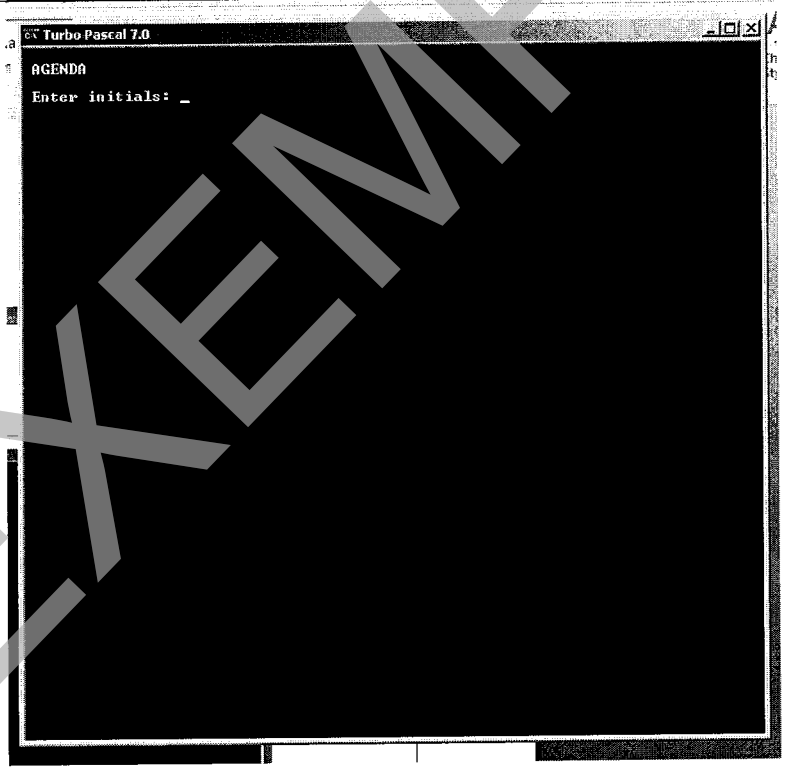


The system can generate statistics "on the fly" – ie using real data, drawn directly from the files generated by the system.

38



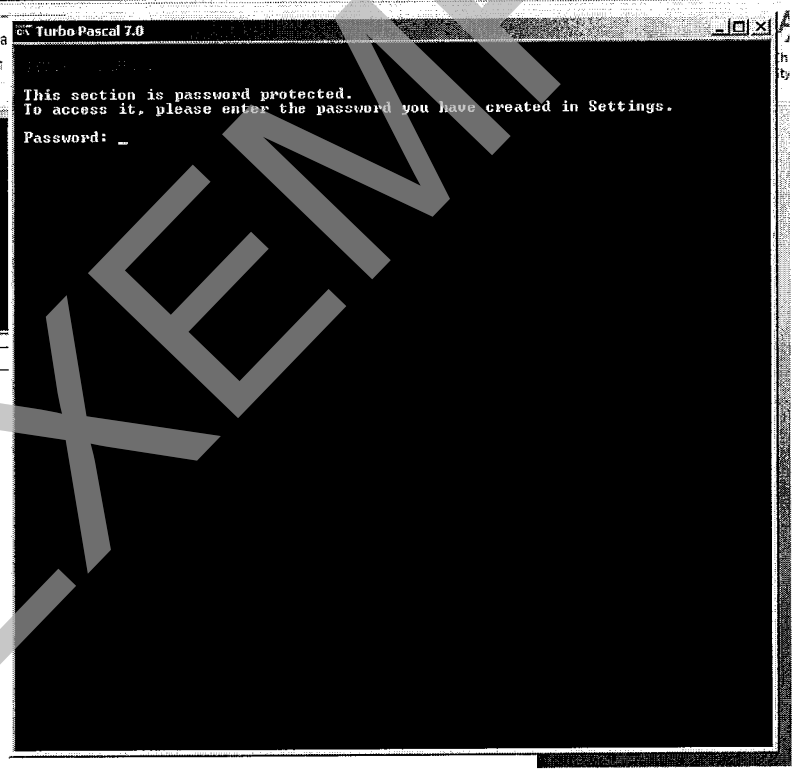
Before showing an agenda, the system should prompt the user for a technician's initials. By doing so, the system can create a personalised agenda.

39	 The system prompts the user for a technician's initals, but the technician has no requests assigned. The system returns the user to the main menu.
40	 The system prompts the user for a technician's initals, but the technician cannot be found. The system returns the user to the main menu.

41

The system to checks to see if a password exists. If it does, the user is asked for the existing password. Otherwise, they are asked to creat eone.

42



When a password already exists, the user is prompted for the existing password before the system will allow them to change the existing password.

43	 The system asks for a password and will keep asking until the correct password is provided.

No separate contents page.

God introduction.

USER GUIDE

INTRODUCTION

Lots of people asking you to just fix this thing, or another? "Sure", you say – before realising that all this is going to take time and it's going to be a nightmare to organise. Step forward the Job Management System, or JMS, for short… a great way to organise where you're up to with job requests.

Simple Provided you can use a keyboard, and have a grasp of the English language, you're all set to start using JMS straight out-of-the-box.

Cross-Platform You can install JMS on almost any computer that will run some form of Pascal compiler – be that a Windows box, a Mac or a Unix machine.

Cost Effective When you purchase JMS, you're not just purchasing the software to use from an end-user's perspective… throw the program at a developer and see what they can come up with! JMS includes both the program itself, the file structure required to use JMS, and the source code.

JMS is a complete all-in-one solution for organising a job request from start to finish, and tracking it's progress all the way along.

INSTALLATION

Very detailed installation instructions.

System Requirements

To run the system as intended, you will require:

Windows

- A fully-working, installed copy of any version of Windows capable of running a program in MS-DOS, including:
  - Windows 95
  - Windows 98
  - Windows ME
  - Windows NT
  - Windows 2000
  - Windows XP
  - Windows Vista
  - Windows 7

> **Note:** *Versions of Windows released before 1995 are not supported due to the program's requirements of a disc tray. Additionally, these operating systems are categorised as "legacy", and little support is provided for them.*

- A Pascal compiler – the system was built using Turbo Pascal 7, but any Pascal compiler capable of running units should work.
- A maximum of 5mb hard disk space, and correct user permissions to access this space and save files to it.
- A disc drive capable of reading from CD-ROM.

Mac OS

- A fully-working, installed copy of Mac OS 9 or above.
- A Pascal compiler, such as FreePascal or GNU Pascal for Mac
- A maximum of 5mb hard disk space, and correct user permissions to access this space and save files to it.
- A disc drive capable of reading from CD-ROM.

Linux

- A fully-working, installed copy of a Linux distribution
- A Pascal compiler, such as GNU Pascal
- A maximum of 5mb hard disk space, and correct user permissions to access this space and save files to it.

- A disc drive capable of reading from CD-ROM.

Installation for Windows machines

1. Insert the Installation Disc into your computer's disc tray, then navigate to Start, followed by My Computer (or Computer on machines running Windows Vista or later).

2. Double-click the disc tray – the disc itself is called "JMS-INSTALL" – and navigate through Windows Explorer to the "install" folder.

3. Open a new Explorer window, by opening up My Computer (or Computer on machines running Windows Vista or later). Navigate to:

   - The root drive where Windows is installed: usually C:
   - Documents and Settings
   - All Users
   - Start Menu

4. Create a new folder in this directory called "JMS". Drag the contents of the "install" folder on the disc into this folder.

5. Once the files have copied across, restart your machine. When your computer reboots, you should see the JMS files in the Start Menu. These files should be available to anyone who has a user account on the machine you use.

> **Note:** *If you only wish to allow certain users to access the system, simply navigate to Root:/Documents and Settings/User/Start Menu, where User is the name of that user's account. You will need to do this for each user on your system who requires access.*

> **Note:** *.pas files must be associated with a Pascal compiler before you run JMS. In Windows XP, Vista and 7, you can use Program Access and Defaults in Control Panel. For earlier versions of Windows, you may need to edit the Registry using regedit.*

Installation on Mac OS systems

1. Insert the Installation Disc into your computer's disc tray. For systems running Mac OS X, the disc will appear on your desktop. For earlier systems, open a new Finder window and navigate to the "JMS-INSTALL" disc, which will appear on the left.

2. Navigate to the Applications folder in a new Finder window. For those running more modern versions of Mac OS, the "Go" menu at the top of Finder may be of use here. Create a new folder within Applications called "JMS".

3. Switch back to the Finder window containing the disc's directories. Copy the files from within the "install" directory to the JMS directory.

4. Close the two Finder windows. You will now find JMS installed as an application. As long as .pas files are associated with a Pascal compiler, you should have no problems using the system. If .pas is not associated with a compiler, use System Preferences to associate it with your compiler.

Installation on Linux systems

Installation on Linux systems varies largely regarding which GUI system you are using. General instructions for installing programs can be found on the internet, and in the large GNU and Unix communities which exist.

Satisfactory instructions for many but not all aspects of system.

Add New Request

```
ADD REQUEST

Forename:                   GEORGE
Surname:                    WASHINGTON
Department:                 POLITICS
System Name:                LH-01
System Location:            LH
Category:                   PROJECTOR
Description:                Projector will not output feed.
```

1. From the main menu, use the numerical keys on your keyboard to navigate to the Requests menu – option 1 – then Add Request – option 1.

2. The system will ask for various pieces of information relating to the request: a full list is included below. After entering information for each question, press [ENTER] to save the information to the request's file.

    - Forename
      The forename of the user requesting the service.

    - Surname
      The surname of the user requesting the service.

    - Department
      The department which the user belongs to – eg Chemistry, Maths etc.

    - Today's Date
      The date is generated automatically from the system clock: no changes need to be made.

    - System Location
      The location of the system in terms of the room it is located in – for example, LK, LL, UA, UI etc.

- System Name
  The name of the system – generally, this will be the computer's name on the network – for example, LK-01, LL-02 etc.

- Fault Category
  The category which this fault belongs in: urgent, high, moderate, low. It will be up to the user to decide how important each request is.

- Fault Description
  An in-depth description of the fault.

3. After entering each piece of information, the system will produce a summary screen, and you will be allocated with a numerical tracking reference. You can use this tracking reference to update the status of your request and to track the job as it progresses through to being completed.

Tracking a Request

```
TRACKING

This service allows you to check the status of any open or archived requests.
You can view:
      - Request details
      - Status updates
      - Technician's notes

A tracking reference is required to use this service.
Tracking references are available once you have submitted a request.
It is NOT possible to relocate a tracking reference if it is misplaced.

Tracking Reference:

To quit, press 0 on the numpad, then hit [ENTER].
```

1. From the main menu, choose Tracking (option 6) using the numerical buttons on your keyboard.

2. A brief description is included at the top of the tracking facility. Below this, the cursor prompts you for the tracking reference of a request. Enter this tracking reference, and press [ENTER] to view information about it.

3. The system will then ask you whether you wish to:

    a) Display the notes and status of the job

    Displaying the notes and status of a job will pull information from various files in use elsewhere in the system. You will be able to see the current status of the request, which technician it is assigned to and any notes entered by the technician assigned to the request.

    b) Display the original details associated with the request.

Note:	*Some information may appear incomplete. This is due to the relevant information not being completed in other parts of the system. As soon as this information is complete, the tracking page for this request will be updated.*

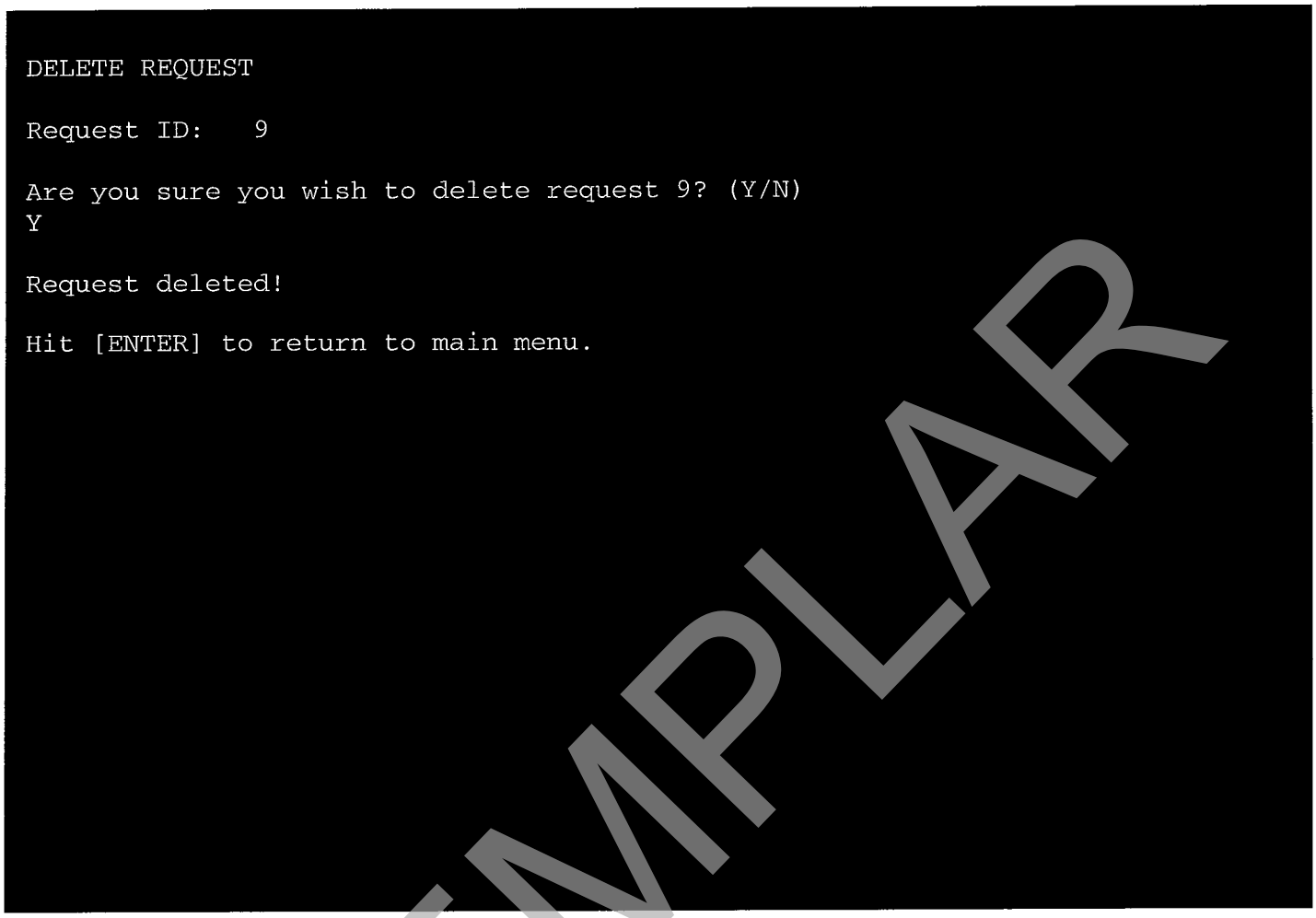
Note:	*No current information will be supplied with this type of tracking request: only information available at the time of the request being added to the system will be shown.*

Update Request

To update the status of a request:

1. From the main menu, choose option 1 – Requests.
2. From the Requests sub menu, choose Option 2 – Update Request.
3. The system will ask you for the request's ID. Enter this here.
4. If the request has no technician assigned, you'll be asked to enter a technician's initials here. If a technician has already been assigned, this step will not appear.
5. You will be presented with a list of available status updates – including an Other option for when the preset updates do not match the actual status of a request.
6. Next, you'll be asked if you want to add notes to the request. Hit Y for Yes, and N for no. There's a 255 character limit on notes.
7. Finally, you'll be presented with a summary screen of the details you entered. Hit Enter to return the main menu. Your update has been saved accordingly.

Delete Request



From the main menu, choose the Tracking option and follow the instructions.

Tracking allows you to access request details, status updates and technician's notes, where applicable, for each request.

Agenda

```
AGENDA for ABC

ID    Date        St    Name        Location          Notes
1     14/04       1     MEAKIN, P   LL (LL-01)        Cannot connect to network.
2     16/04       3     LEE, D      UN (UN-94)        Cannot print. Awaiting ink.

Hit [ENTER] to return to main menu.
```

1. From the main menu, choose Agenda (option 5).
2. Enter the initials of the technician whose agenda you would like to view.
3. Details of each request are displayed on one line, under various headings:
    - Request ID
    - Date
    - Current Status
    - User's name
    - Location
    - System Name
    - Notes

STATISTICS

```
STATISTICS

Currently, there are 37 requests in the system, of which:
      - 9 jobs are deemed open
      - 8 jobs are under investigation
      - 4 jobs are awaiting parts
      - 6 jobs are being resolved
      - 7 jobs require attention
      - 3 jobs are ready to be collected
      - 0 jobs are deemed as other

- ABC has 6 jobs assigned.
- DEF has 8 jobs assigned.
- GHI has 7 jobs assigned.
- JKL has 9 jobs assigned.
- MNO has 4 jobs assigned.
- PQR has 3 jobs assigned.

The last request was submitted on 14/04/10.
```


As a request progresses through the system, the statistics feature collects and processes data. This information is then easily accessible to a user from the main menu, by choosing the Statistics option. As the statistics collected could be seen as "sensitive" data, users are asked for the password set from within the Settings screen before they can view them.

To access Statistics:

1. From the main menu, choose the Statistics option.
2. You will be prompted for your password. Enter this here.
3. Statistics will be shown by the system.

Assets

```
ADD ASSET

Assets can be one of six types:
      1 Workstation
      2 Server
      3 Projector
      4 Whiteboard
      5 Peripheral
      6 Other

Asset Type:
Asset Manufacturer:
Asset Model:
Asset Location:
Purchase Retailer:
Purchase Date:
Warranty Expires:
```

From the Assets submenu – option 2 – choose Add Asset – option 1 – then enter the information requested,

Additionally, the assets menu also provides you with ways to find assets and to delete assets. These features work in much the same way that the features for viewing requests and for delete requests work.

Troubleshooting

Screen	Problem	Cause	Solution
Add Request	The wrong date is being shown in the "Today's Date" field	The date for the "today's date" field is gathered automatically from your computer's system clock.	Adjust your computer's clock accordingly.
View Request	No information is being shown when I enter a request's ID.	There is no request associated with this ID.	Check that you have entered the correct ID.
	I have used the Find Request by Surname feature. When I hit enter to return to the main menu after viewing request details, I am presented with another set of results.	In this case, the surname you entered is shared between more than one request.	After the last result has been displayed, hitting Enter will return you to the main menu.
Update Request	When I enter a request's ID, I am returned to the Requests submenu.	There is no request associated with the ID you entered.	Check that you have entered the correct ID.
Agenda	"This technician does not exist" message.	The technician you entered does not have any assigned jobs.	Check that the technician you have entered has jobs assigned to them.
Statistics	Some statistics appear blank.	This occurs when certain features have not been used.	Make sure to use all of the features to update a request.
Delete Request	This request has been deleted.	This is normal – it occurs when the request you've asked the system to delete has been deleted.	This is normal behaviour.

Good section on error handling.

BACKUP

It's really important to keep a backup of the files the system generates if you wish to preserve data integrity in cases where data loss occurs. You can schedule your computer to do this automatically. On a Windows computer:

1. Open Notepad – notepad.exe from a command prompt or the Run dialog.
2. Type in the following code*:

```
xcopy /e /v /y C:\openreq.dat E:\jms\backups\openreq.dat
xcopy /e /v /y C:\iddata.dat E:\jms\backups\iddata.dat
xcopy /e /v /y C:\assets.dat E:\jms\backups\assets.dat
xcopy /e /v /y C:\employees.dat E:\jms\backups\employees.dat
```

Note: This batch file uses the following switches:
- /e Copies all data within file
- /v Verifies data has been copied successfully
- /y Automatically overwrites old data without prompt

Feel free to change switches as you see fit. More information on switches is available online.

This batch file also assumes your files are stored directly in C:\ and your backup drive is at E:\. Feel free to change the directories as you see fit. You must always point to the exact file – XCopy cannot make the files without you declaring their name and type, as shown above.

3. Save the file as jmsbackup.bat in the root directory where all other system files exist.
4. Find Scheduled Tasks – usually in Control Panel. Add a new scheduled task, declaring the following:

Application: Browse to location of jmsbackup.bat
Name: A useful, descriptive name – eg JMS Backup
Perform: Daily
Start time: A convenient start time, at the end of the day
Start date: Default date
User name: User name of account on computer with administrator access
Password: Password for above account

RECOVERY

If you suffer loss of data, you can recover any files backed up using the above method by:

1. Open Notepad – notepad.exe from a command prompt or the Run dialog.
2. Type the following code:

```
xcopy /v E:\jms\backups\openreq.dat C:\openreq.dat
xcopy /v E:\jms\backups\iddata.dat C:\iddata.dat
xcopy /v E:\jms\backups\assets.dat C:\assets.dat
xcopy /v E:\jms\backups\employees.dat C:\employees.dat
```

Note: This batch file uses the /v switch to verify that data copied from a backup to the working root directory is copied across successfully, and without errors. Feel free to change switches as you see fit. More information on switches is available online.

This batch file also assumes your files are stored directly in C:\ and your backup drive is at E:\. Feel free to change the directories as you see fit. You must always point to the exact file – xCopy cannot recover the files without you declaring their name and filetype, as shown above.

3. Save the file as jmsrecovery.bat in the root directory of the system.
4. Open jmsrecovery.bat when you need to recover data from backed-up files.

Appraisal

Project Performance

Objective 1: *The system must allow certain data about each job request to be stored. At the very least, the system needs to be able to store the same data that is currently collected under the existing system, unless there is a good reason to not store the data.*

This objective has been met completely. When adding a new request, the system asks the user for the same information that has been recorded on paper, and also adds a few new fields.

Objectives 2a and 2b: *The system needs to be able to show the current status of a job upon request. This information should be accessible from the Requests menu and should be shown within the Tracking feature.*

This objective has, again, been met completely. Users can access information, including the current status of a job through the Requests menu, which is directly accessible from the main menu. In addition, the user can also access the current status of a job when job information is viewed from within the Tracking feature. Both of these ways to view the job status are easy to access.

Objective 3: *The system needs to be easy-to-use and logical for a user to follow.*

The system takes advantage of a piece of hardware connected to the vast majority of computers, and which – if not already connected – can be added to most computers: the keyboard. As the keyboard is one of the major components that users have access to, it is expected that if a person can use a computer, they can use a keyboard. To this end, to navigate through various menus and enter data into the system, the numerical keys on the keyboard are used.

Objective 3a: *A basic colour scheme should be followed throughout the system.*

A structured colour scheme is important for several reasons. Firstly, it is aesthetically pleasing to users: if a system is aesthetically pleasing, a user is more likely to want to use the system for the purpose it is intended for. Secondly, the system's colour scheme makes use of several basic colours – such as black backgrounds, light grey text, yellow text for warnings and red text for errors and serious messages. As a result, the user can get used to the colour scheme used so that they can instantly recognise when an error has occurred.

Objective 3b: *Wherever possible, the system should make use of widely-recognised keyboard shortcuts. For example, the numeric key for "3" should not delete an asset in one feature but confirm an option in another feature with good reason for doing so.*

By using common keyboard shortcuts – for example, Y and N to confirm choices at a yes/no prompt – the system instantly appeals to users. From a user's perspective, this makes the system more accessible, and hopefully, they will be more inclined to use it. If I were to design the keyboard options in a way which was not as obvious, I would risk alienating users as they would undoubtedly find the system confusing and too complex.

Objective 4: *The system should work on as many platforms as possible.*

The system has been created in the Pascal programming language. As long as a user has a working computer, and can find a Pascal compiler for their operating system – of which there are plenty for Windows, Mac, Unix and other operating systems – then they can run the system as it was meant to be run.

Objective 4a: *There should be no extensive requirements to heavily adapt any existing systems to run this system as intended.*

Apart from the requirement of a computer capable of running one of a number of widely-available Pascal compilers, my program does not require any specialist hardware or software to be installed and integrated with current systems. The system was designed to be run "out-of-the-box" as far as possible. I believe that the setup procedure is a quick and easily completed one, and that the backup and recovery code provides a high degree of customisation for organisations implementing my system.

Objective 4b: *To achieve Objective 4a, it should also be noted that the system should be built to run on as many operating systems as possible.*

Where a Pascal compiler is available for the operating system a user is running, the system will be able to run on the user's computer. There are many compilers available for Windows, Mac OS and Linux machines, as well as some built-in support for systems running Solaris, BSD and OS/2.

Objective 5: *The system needs to produce statistics about job requests.*

The system can produce various statistics about jobs, based on procedures which gather and process data when new requests are created. Examples of statistics provided include a breakdown of all requests within the system by status, and a breakdown of requests by technician so that it is easy to see which technicians have too much work on their hands.

Objective 5a: *The system should be able to show the total number of jobs and assets.*

The system can show the total number of jobs and assets that have been added to the system. In addition, the system will also break down the number of jobs and assets by type. Jobs are broken down by their current status, whereas assets are broken down by their type – for example, whiteboard, projector or server.

Objective 5b: *The system should be able to show the total number of jobs assigned to each technician.*

The Statistics screen can display the total number of jobs assigned to each technician. This total includes all of the jobs included in the system, regardless of whether they have had their status updated or not since they were first created.

Objective 6a: *All files generated by the system should be able to be backed up with minimal intervention from the user.*

The system uses a batch file to automate the backup procedure. Through the batch file, the user is able to specify what exactly needs to be backed up, and where it should be backed up to, as well as any other arguments to do with verifying data. Once the batch files for backup and recovery are created and the host computer has been set up to automatically run the files at an allocated time, the user does not need to intervene with the backup: it will happen automatically. Whenever a user wants to update where the files are updated to, they can simply edit the batch file.

Objective 6b: *Security is important. As such, a means of password-protecting certain features should be implemented.*

The system allows the user to set a password which can be used to protect features providing output which could be described as sensitive. Examples include the Statistics feature, and the ability to clear all data stored within files. This level of security – whilst not foolproof – does go a long way to preventing accidental and malicious destruction of data.

Good comparison to objectives, with decent explanations.

Possible Extensions

- It would be useful if the system could be used on multiple computers at once. This could allow all technicians to have the system running in the background on their computers, and would allow more than one job to be processed at once. Additionally, if a computer cannot be accessed, data could still be processed using the system.

- The system could be made more useful if the ability to create relationships between employees, requests and assets was available. For example, if you could link assets to employees, you could then only accept requests from employees who have access to a particular asset.

- Being able to choose which features are password protected without having to edit the program's source code would be useful. For some users, for example, statistics may be sensitive: however, for others, they could be used as motivational tools.

Sensible ideas.

User Feedback

I asked three different people to use my system to see how effective they found it, and to score it out of five in a number of different areas. The following results are averages of the individual results.

Area	Score
Layout and design	4.6 (5, 4, 5)
Speed and responsiveness	4.3 (4, 5, 4)
Ease of use	5.0 (5, 5, 5)
Level of detail of questions asked	4.3 (5, 4, 4)
Accuracy of data pulled from files	5.0 (5, 5, 5)
Average Score:	4.64

Additionally, I asked for a few sentences from each of the people that tested the system to describe how they thought it worked overall:

"Overall, I think the system works very well. It records the information accurately and I think I could use the system instead of the paper-based system we currently use in the office. After installation has been completed, and the system has been set up to automatically backup the data it collects and stores in files, I thought the system was a pleasure to use – and it was not resource-intensive, so I could use other programs at the same time." – local estate agent

Why?

Analysis.

I wanted my program to appeal to a wide range of users: not just those within a school environment. If I were to sell my product, my core market would be very limited if I were selling only to educational establishments. To overcome this, I asked a local estate agent to play with my system. Personally, I am extremely grateful for the comments he has provided, and I think that having an end-user tell me that my system works "very well", and that it's a "pleasure to use" goes a long way to showing how end-users with less technical experience than IT technicians should have can still understand the logic behind my program.

"Despite my initial bugbears at having to install one program [a Pascal compiler] *to be able to run another system, I did find the program easy-to-use and responsive. Additionally, I was impressed by the level of security built into the program: not only could I set a password to protect data, but I could also implement a backup system to copy the files created by the system to an external backup drive. By changing the backup drive's file path, I could also backup to a mapped file server."* – IT technician

Although this user was sceptical about the effectiveness of the program considering it relied heavily upon another program, I am glad that he found the system "responsive" and "easy-to-use". I am especially pleased with the comments he provided about the "level of security built into the program". Security was a major concern when I was building the program: of course, I do not expect the security features within my program to be entirely foolproof. The features were included as deterrents, and I think that they will serve their purpose of deterring people from accessing information which they shouldn't access.

"It works well. I especially like the way that I can choose to run the program in full-screen mode or in window mode, by using the Alt-Enter shortcut on my keyboard. It's rather useful as a quick way to record requests that comes in from users. Having the ability to host the

– IT Technicians Jobs

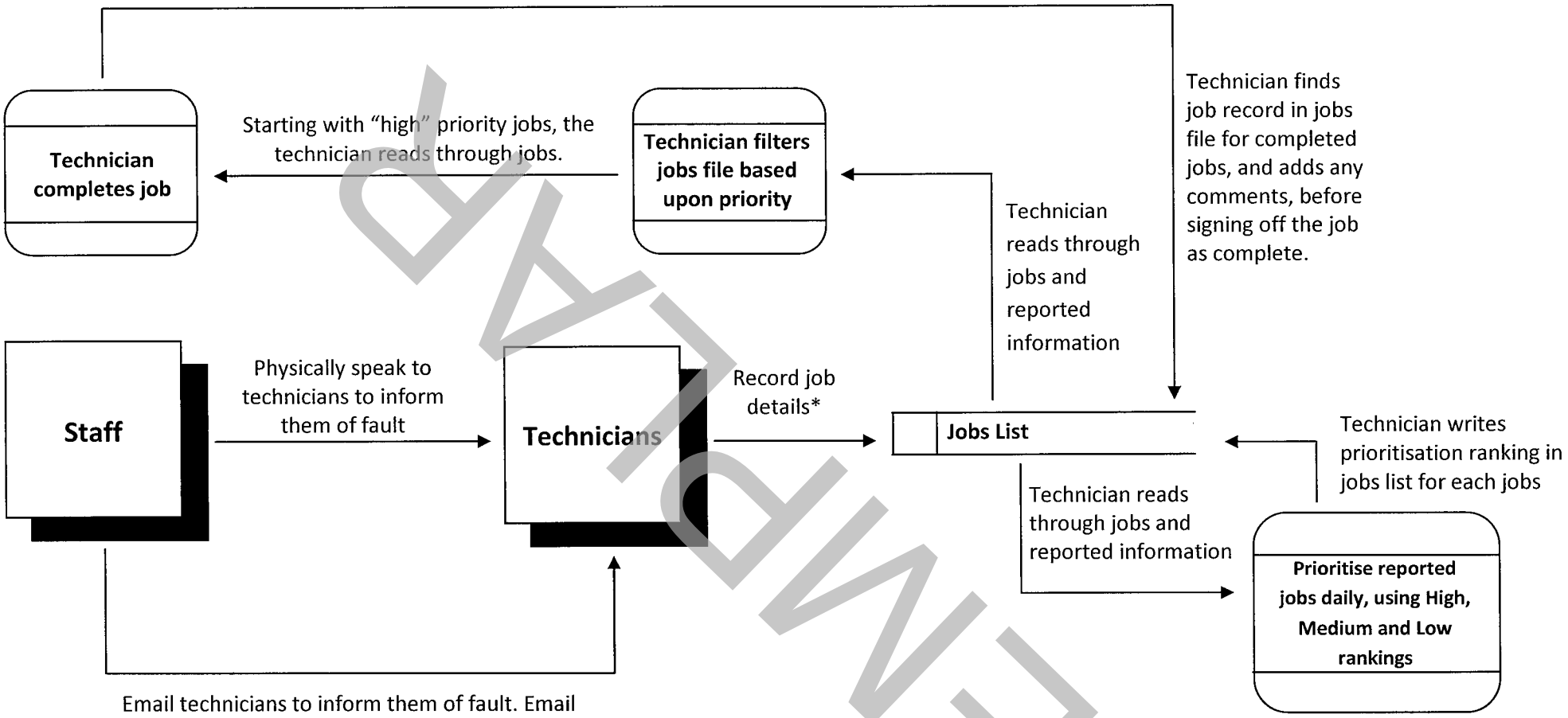
system on a network, however, would have been useful. At the moment, the ability to change the file path for backups and restoration is useful, so that I can backup files along with all other backups to be made." – IT technician

To begin with, this user has pointed out a slight limitation with the program. Unfortunately, on computers running Windows Vista and later, it is not possible to run the program in full-screen due to the design of the operating system. I like how the user notes the program is "quick" and "useful", and am grateful that he has provided a possible extension to the program – which I have noted above – by suggesting that it would be useful to "host the system on a network". I am also glad that the user found the backup system to be "useful", and that my system integrates with the backup server he already has in place.

– IT Technicians Jobs

Sample of Measures Currently Taken to Store Data

NAME	DEPARTMENT	DATE	SYSTEM	COMMENTS	PRIORITY

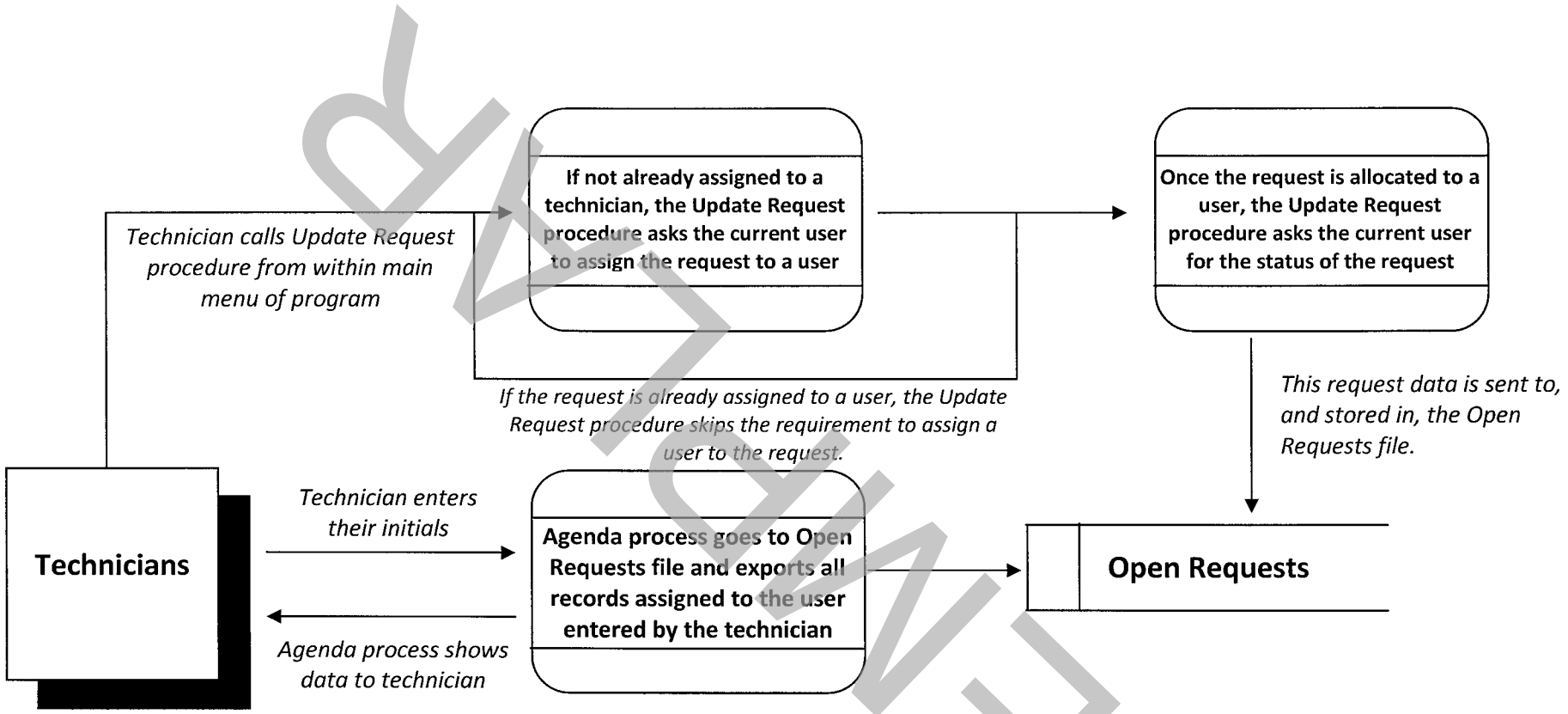


* Details recorded for each job include the name of the person who originally reported the error, as well as the department they belong to. The system name and location, and a rough job description are also noted.

A2

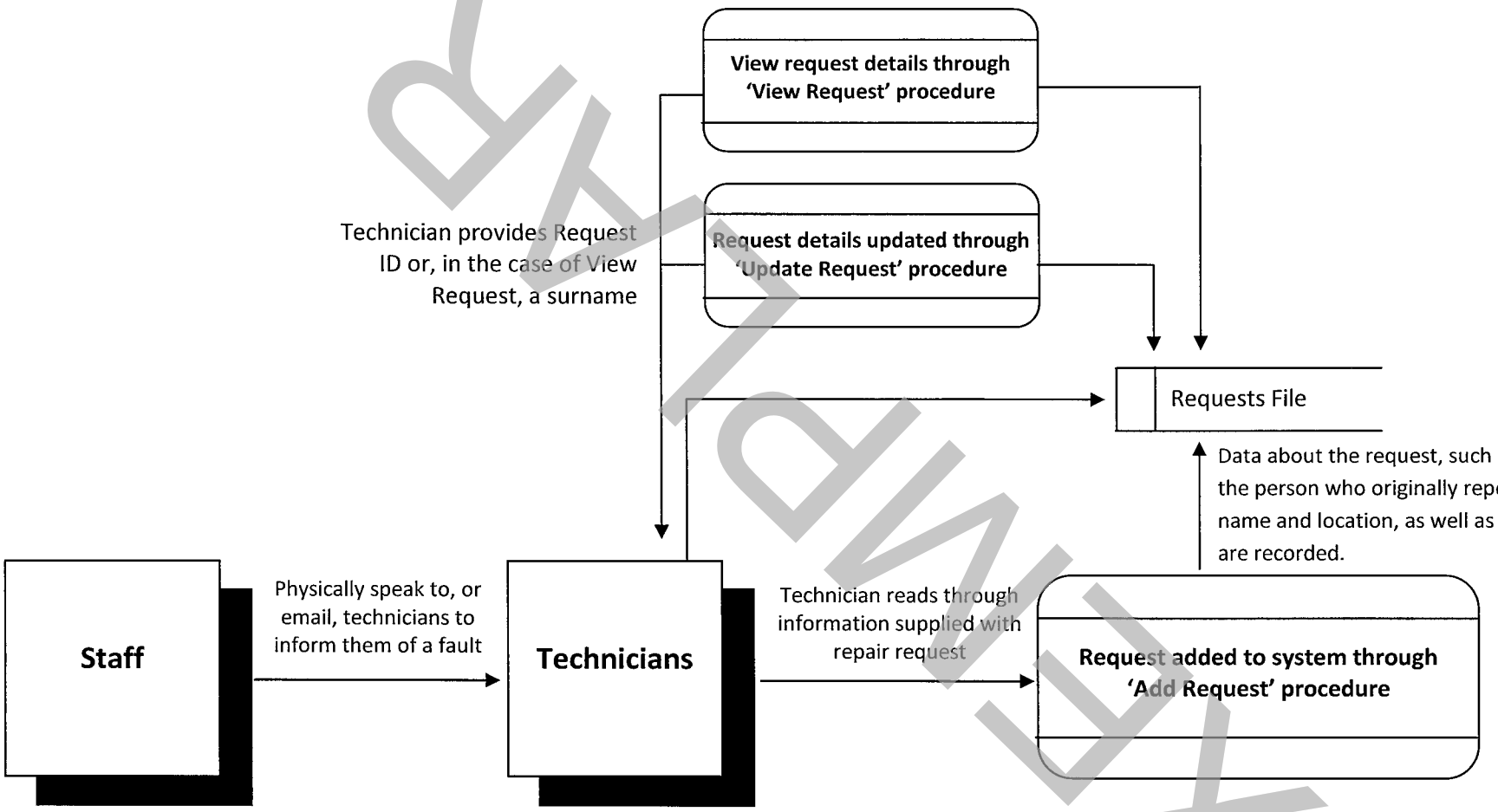
Data Flow Diagram for Existing System

Good DFDs for existing and proposed systems.



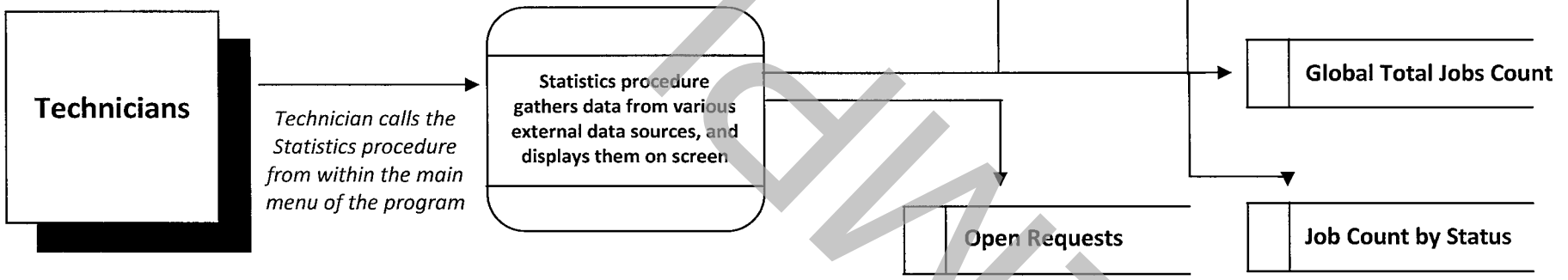
Data Flow Diagram for Proposed System
Agenda procedure

A3



Data Flow Diagram for Proposed System
Request procedures and functions

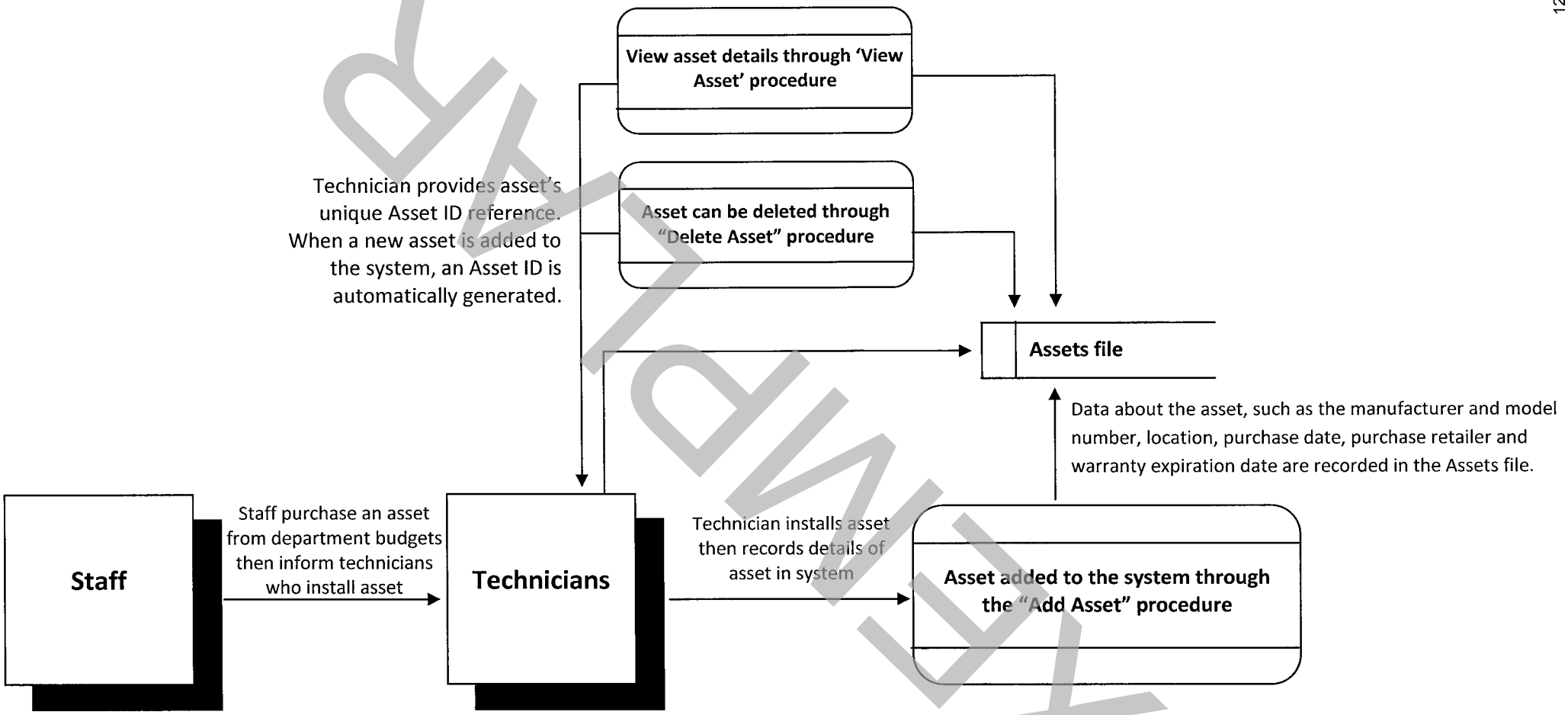
A4



Data Flow Diagram for Proposed System

Statistics procedure

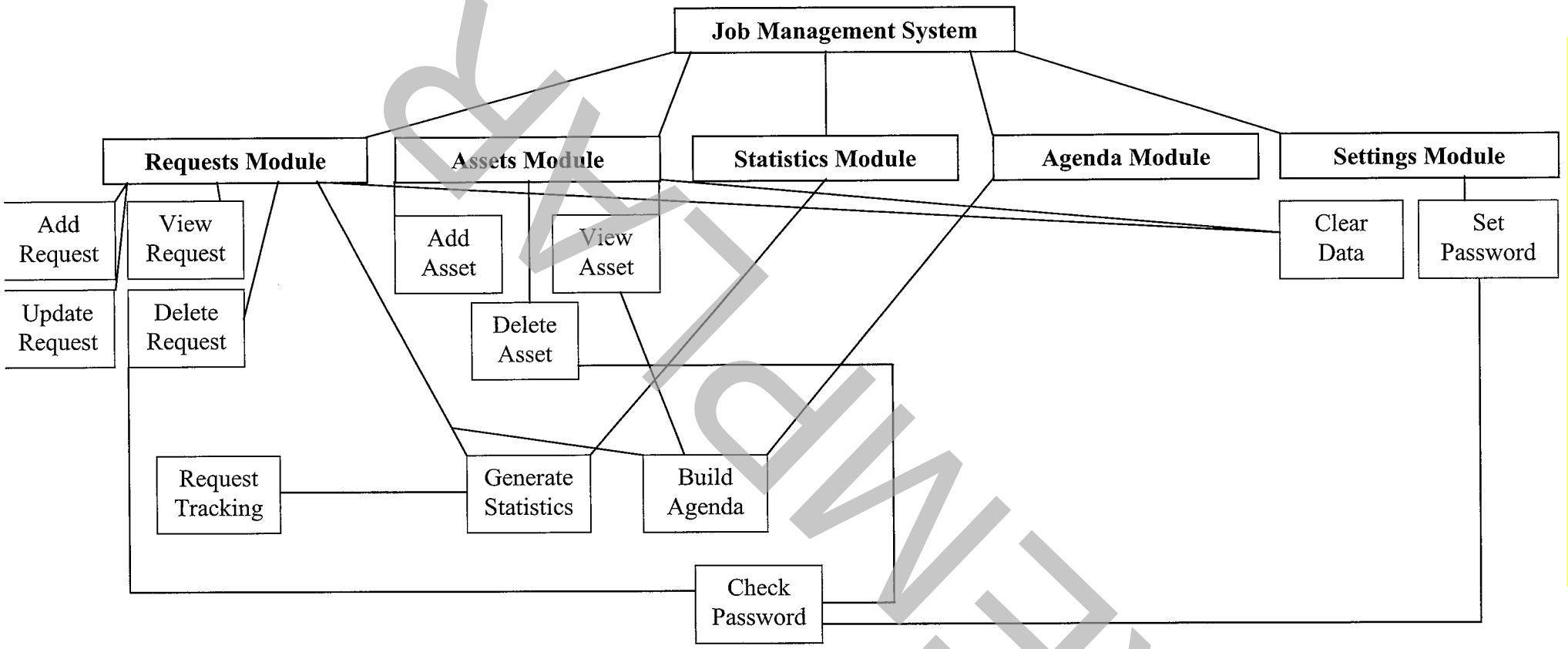
A5



Data Flow Diagram for Proposed System

Assets procedures and functions

A6



A satisfactory design, but a bit confused?

A7

DESCRIPTION OF MODULAR STRUCTURE OF SYSTEM

Sample of Planned Data Capture and Entry

Forename:	
Surname:	
Department:	
Today's Date:	☐☐ /☐☐ /☐☐☐☐ **DD/MM/YYYY format**
System Location:	
System Name:	
Fault Category:	
Fault Description:	

Sample of Planned Valid Output

Request ID: 65
Forename: J
Surname: E
Department: Maths
Submission Date: 13/03/2010
System Location: UF
System Name: MT01
Fault Category: Printer
Fault Description: Low ink